VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Asistenční služba pro cestující

Traveller Assistance Services

2013 Tomáš Dobeš

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Tomáš Dobeš**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Asistenční služba pro cestující
Traveller Assistance Services

Zásady pro vypracování:

Cílem diplomové práce je vyvinout aplikaci pro platformu Windows Phone 7 poskytující komplexní asistenční služby pro potřeby cestujících. Hlavní službou je vyhledávání vlakových spojení (s možností rozšíření i na další druhy dopravy jako autobus, MHD, letadlo...), doplněná o možnost zakoupit si lístek či místenku, s podporou zobrazení polohy vlaku na mapě včetně zpoždění a řazení vlaku. Pozice zjištěná z GPS poskytuje informace o nejbližším nádraží a popřípadě také o zajímavostech v daném městě či lokalitě.

1. Server bude implementován v prostředí C# .NET Framework.
2. Ukládání dat bude zabezpečeno v cloudu pomocí databáze Microsoft SQL Azure.
3. Klient bude naimplementován pro mobilní platformu Windows Phone 7.
4. Aplikace bude podporovat zobrazování informací svázaných s aktuální GPS pozici.
5. Součástí bude algoritmus pro inteligentní aktualizaci lokální databáze.
6. Testování aplikace (publikace na Marketplace) a zhodnocení dosažených výsledků.

Seznam doporučené odborné literatury:

[1] Brunetti, R., Windows Azure Step by Step, Microsoft Press, 2011, ISBN 978-0735649729
[2] Klein, S., Pro SQL Azure, Apress; 1st edition, 2010, ISBN 978-1430229612
[3] Troelsen, A., Pro C# 2010 and the .NET 4 Platform, Apress; 5th edition, 2010, ISBN 978-1430225492

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Mgr.Ing. Michal Krumnikl**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 24. dubna 2013 .....

Rád bych na tomto místě poděkoval Mgr. Ing. Michalu Krumniklovi za jeho vstřícný přístup a rady, které mi pomohly při tvorbě této práce.

Abstrakt

Cílem diplomové práce bylo navrhnout a vyvinout aplikaci pro platformu Windows Phone 7 poskytující komplexní asistenční služby pro potřeby cestujících.

První část diplomové práce seznamuje čtenáře s existujícími vyhledávači vlakových spojení pro různé platformy, nastiňuje problémy týkající se vyhledávání cest v dynamicky se měnících grafech a definuje algoritmy, ze kterých čerpá vyhledávání v Asistenční službě a také těch, které jsou na tento typ problému běžně nasazovány.

Ve druhé části se práce zabývá návrhem a implementací aplikace včetně vývoje vyhledávání v jednotlivých etapách. V závěru jsou demonstrovány výsledky provedených testů.

Klíčová slova: Windows Phone, vyhledávání vlakových spojení

Abstract

The aim of this thesis was to design and develop an application for Windows Phone 7 platform providing comprehensive assistance to the needs of passengers.

The first part introduces the reader to the existing train connection search applications for various platforms, outlines the problems of path search in dynamically changing graphs and defines algorithms, which draws from search assistance services, as well as those that are on this type of problem commonly deployed.

In the second part of the thesis deals with design and implementation of applications including the development of search at each stage. The conclusion demonstrated by the results of the tests.

Keywords: Windows Phone, train connection searching

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
ASP	–	Active Server Pages
ČD	–	České dráhy, a.s.
EC	–	Vlak vyšší kvality kategorie EuroCity
EN	–	Noční vlak vyšší kvality kategorie EuroNight
Ex	–	Vlak vyšší kvality kategorie Expres
GPS	–	Global Positioning System
GPX	–	GPS eXchange Format
HTML	–	HyperText Markup Language
IC	–	Vlak vyšší kvality kategorie InterCity
IDOS	–	Informační dopravní systém
LINQ	–	Language Integrated Query
MHD	–	městská hromadná doprava
Os	–	osobní vlak
POI	–	Point of interest
R	–	rychlík
SLA	–	Service-Level Agreement
SMS	–	Short message service
SOAP	–	Simple Object Access Protocol
Sp	–	spěšný vlak
SQL	–	Structured Query Language
SŽDC	–	Správa železniční dopravní cesty
WCF	–	Windows Communication Foundation
WGS	–	World Geodetic System
WMM	–	Windows Mobile Mania
XAML	–	Extensible Application Markup Language
XML	–	Extensible Markup Language
žst.	–	železniční stanice

Obsah

1 Úvod **9**
1.1 Obsah práce 9

2 Konkurenční aplikace **11**
2.1 WMM Jízdní řády 11
2.2 jVlaky 11
2.3 CGTransit 12
2.4 Pubtran 12
2.5 IDOS 12
2.6 IDOS do kapsy 12
2.7 Srovnání českých vyhledávačů 13

3 Vyhledávání spojů **15**
3.1 Problém vyhledávání 15
3.2 Existující algoritmy 16

4 Návrh aplikace asistenční služby **21**
4.1 Motivace a výhody použití asistenční služby 21
4.2 Uživatelé asistenční služby 21
4.3 Struktura projektu 22
4.4 Návrh databáze 23
4.5 Získávání a zpracovávání dat 23

5 Mobilní klient **27**
5.1 Vyhledávač vlakových spojení 27
5.2 Průvodce cestou 30
5.3 Další funkce dostupné offline 32
5.4 Funkce využívající internet 33
5.5 Práce aplikace v různých stavech životního cyklu 37
5.6 Design tlačítek 38
5.7 Testování mobilního klienta 40

6 Webová část aplikace **45**
6.1 Windows Azure™ 45
6.2 Funkcionalita 45
6.3 Zobrazování na mapě 46
6.4 Aktualizace 46

7 Závěr **49**

8 Reference **51**

Seznam tabulek

1 Srovnání českých mobilních vyhledávačů spojení 14
2 Výsledky testování pro Test case #1 41
3 Hodnotící kritéria srovnávacích testů 43
4 Výsledky srovnávacích testů 43
5 Testování spotřeby baterie 44
6 Dostupnost GPS v jednotlivých vozech a na různých místech v nich 44

Seznam obrázků

1 Schéma komunikace mezi jednotlivými vrstvami systému 22
2 Schéma databáze 24
3 Struktura zápisu o zpoždění na babitron.ic.cz 33
4 Fáze komunikace Tile notifikací 36
5 Schéma životního cyklu aplikace ve Windows Phone 37

Seznam výpisů zdrojového kódu

1 Algoritmus prohledávání napsaný v pseudokódu 16
2 Algoritmus A* (pseudokód) 18
3 Získávání dat přímo z paměti 32
4 Definice grafiky tlačítka 38
5 Fragment logiky tlačítka hlavního menu 39
6 Kód vykreslující pozici stanice či POI bodu 46
7 Zjištění velikosti změněných dat 47

1 Úvod

Od počátku věků je v člověku zakořeněna touha cestovat. Vznik hromadné dopravy a jízdních řádů byl tak jen otázkou času. Rozmach železnic započal v 19. století, kdy byla z Českých Budějovic do Lince postavena první koněspřežná trať a brzy po ní následovaly další. Zpočátku na nich jezdilo pouze malé množství vlaků a jejich jízdní řád jsme mohli najít například v novinách. Mezi známé jízdní řády patří také *Vilímkův jízdní řád republiky Československé*. Některé jeho ročníky jsou dostupné k nahlédnutí na internetu.

Od té doby se situace velmi změnila. Cestování se stalo pro většinu z nás téměř nutností. Prakticky denně se potřebujeme dopravit do zaměstnání, do školy nebo například na nákupy. K těmto účelům mnoho z nás využívá hromadnou dopravu. Často potkávám lidi bezradně těkající očima po jízdních řádech ve snaze najít příslušný spoj, jenž by je přepravil do místa určení. Mnohdy ale jízdní řád na zastávce úplně chybí.

Velké změny však nenastaly pouze v dopravě, nýbrž v podstatě ve všech odvětvích lidské činnosti. Obrovský skok nastal také v elektrotechnice a později i informatice. To, co bylo dříve nemyslitelné, je nyní naprosto běžnou součástí našeho života. Současné telefony se pohodlně vejdou do kapsy a čím dál více plní roli počítačů. Logicky udělaly značný pokrok také mobilní aplikace. Běžné je používání vláken, 3D grafiky i dalších vymožeností, které moderní mobily poskytují.

Mobilní aplikace postupně vytlačují z povědomí své „papírové“ ekvivalenty. Totéž se stalo i u jízdních řádů. Klasické knižní jízdní řády často nahrazují interaktivní vyhledávače spojení, jelikož nabízí mnohem vyšší komfort užívání a také více funkcí. Vyhledávačů vlakových spojení je i na českém trhu několik (s nejznámějšími z nich se můžeme seznámit v kapitole 2).

1.1 Obsah práce

Asistenční služba pro cestující je mobilní offline vyhledávač vlakových spojení. V první části diplomové práce se seznámíme s konkurenčními aplikacemi a porovnáme jejich výhody a nevýhody. Poté si nastíníme problém hledání v dynamicky se měnících grafech. Nadefinujeme si algoritmy, z jejichž principů vychází vyhledávání v Asistenční službě a také ty, které se dnes běžně používají k řešení problému vyhledávání spojení mezi dvěma místy.

Následující část práce pojednává o návrhu a implementaci Asistenční služby s vysvětlením zajímavostí týkajících se tvorby aplikace včetně databázové vrstvy. Čtenář se může dočíst i o rozšiřujících funkcích mobilního klienta včetně vysvětlení jejich principů. Zmíníme také grafický návrh tlačítek hlavního menu za použití značkovacího jazyka XAML a technologie Silverlight.

Dále lze v textu nalézt komentované výsledky různých typů testů, které byly v průběhu vývoje a také při nasazování aplikace provedeny.

Závěr práce patří webové části aplikace. Stručně si vysvětlíme její funkce a využití, zastavíme se u problému propagace nových aktualizací do mobilního klienta a lehce se dotkneme cloud computing technologie Windows Azure™.

2 Konkurenční aplikace

Asistenční služba pro cestující je primárně vyhledávač vlakových spojení. Mobilních aplikací, určených ke stejnému účelu, je na českém trhu několik. V této části práce se seznámíme s těmi nejznámějšími z nich.

2.1 WMM Jízdní řády

Aplikace WMM Jízdní řády od Mikyho WOW je vyhledávač dopravního spojení. K dispozici je autobus, vlak, letadlo a také městská hromadná doprava. Zdrojová data jsou poskytována společností CHAPS, stojící za tvorbou populárního IDOSu. WMM Jízdní řády nabízí také další funkce, jako je zobrazení trasy na mapě, GPS našeptávač zastávek apod. Aplikace je plně kompatibilní s Windows Phone 7 a 8.[1] Hlavním záporem je zjišťování spojení pouze prostřednictvím online služby, což omezuje použitelnost pouze na prostředí s internetovou konektivitou.

Ovládání celé aplikace je velmi jednoduché a snaží se dodržet standardy Windows Phone. Základní rozhraní je rozděleno do několika záložek, mezi nimiž je možno přepínat pomocí vodorovných gest.

První záložka s názvem *Hledat* je vlastní formulář pro specifikaci parametrů vyhledávaného spojení. V horní části zvolíme typ dopravy, výchozí, cílovou stanici a také datum a čas.

Druhá záložka (*Spojení*) obsahuje výsledky vyhledávání. Na tuto kartu nás aplikace sama přepne po stisknutí tlačítka vyhledávání. Je možné se sem kdykoliv dostat vodorovným pohybem prstu po displeji. Výsledky vyhledávání obsahují všechny nejnutnější informace a to včetně odhadování přesunů, celkového času jízdy a případných zpoždění daných spojů. V případě, že potřebujeme o konkrétním spoji zjistit více informací, stačí na něj jednoduše kliknout, čímž se nám zobrazí jeho detail.

Třetí a zároveň poslední záložkou jsou *Oblíbené*. Sem si lze uložit všechny oblíbené trasy, které často používáme. Tyto trasy se automaticky rozdělí do skupin podle dopravních prostředků a u každé trasy se zobrazí také malá ikonka umožňující vyhledávání spojení v opačném směru. Přidávat nové trasy mezi oblíbené je možno pomocí tlačítka v detailu spoje.[2]

2.2 jVlaky

Aplikace jVlaky je vyhledávač vlakových spojení napsaný v jazyce Java, díky čemuž je snadno použitelný zejména u starších mobilních telefonů bez operačního systému. jVlaky byly velmi oblíbenou bezplatnou aplikací. Uživatelé oceňovali snadnost použití a relativně dobré výsledky vyhledávání[3]. Aplikace nabízela také možnost vyhledání polohy vlaku ze dvou různých serverů. V poslední verzi však již příliš nefungovalo.

Vyhledávání bylo možné velmi dobře parametrizovat. Nechyběl ani prohlížeč železničních stanic a jejich spojů, včetně zastávek příslušného vlaku a poznámek o spoji. Aplikace však obsahovala nepříjemné chyby. Občas vyhledávala spojení v opačném směru

(tedy např. namísto Praha hl.n. - Brno hl.n. nalezla spoj Brno hl.n. - Praha hl.n. se zápornou dobou jízdy). jVlaky byly vydány naposledy v prosinci roku 2011 a dále se už, podle autora zejména kvůli nástupu chytrých telefonů, nevyvíjí[4].

2.3 CGTransit

Konkurenční aplikace pro operační systémy Windows Phone, Android a iOS. CG Transit je aplikace umožňující offline vyhledávání[1] dopravních spojení v jízdních řádech České republiky a na Slovensku. Aplikace legálně používá oficiální data jízdních řádů z portálu IDOS, jemuž se svojí funkčností podobá, je ale plně přizpůsobena možnostem platformy Android. Vyhledávání dopravních spojení probíhá kompletně offline, takže není nutný žádný datový tarif. Před použitím jízdního řádu je třeba zaplatit jednorázový poplatek a kompletní data stáhnout do mobilního zařízení. [5, 6]. Na Windows Phone její vývoj začal oproti ostatním platformám později, a proto nejsou některé funkce ještě k dispozici.

2.4 Pubtran

Pubtran nabízí vyhledávání vlakových spojení prostřednictím dat IDOSu. Uživatel může využít možnosti uložení spojení do kalendáře, sdílení e-mailem či SMS zprávou a také inteligentního našeptávače stanic. Aplikace umožňuje řazení stanic podle vzdálenosti a četnosti použití, samozřejmostí je zobrazení dráhy spoje prostřednictvím Google map. Aplikace nabízí jednoduché rozhraní. Podle recenzí [7] je největším problémem aplikace komunikace s portálem IDOS. Pubtran přestavuje online alternativu k aplikaci CGTransit.

2.5 IDOS

Ačkoliv se nejedná o mobilní aplikaci, nelze na ni v tomto výčtu zapomenout. IDOS je nejpopulárnější webová aplikace určená k vyhledávání spojení včetně různých kombinací. Každé vyhledávání lze také parametrizovat. Tuto aplikaci poskytuje oficiální poskytovatel dat jízdních řádů - firma CHAPS. IDOS zvládá navigaci po celé Evropě, včetně mezinárodních vlaků, mezistátních autobusů nebo letenek.

2.6 IDOS do kapsy

Mobilní verze plnohodnotného IDOSu pro iOS. Umožňuje snadné vyhledávání v jízdních řádech vlaků, letadel a MHD více než 60 měst ČR. Již několik let patří k nejoblíbenějším českým aplikacím pro tuto platformu.[8]

Narozdíl od některých konkurenčních aplikací, kde je nutno platit za licence, IDOS do kapsy nabízí kompletní on-line přístup k veškerým datům, která jsou neustále aktuální. Pokud však víme předem, že nebudeme disponovat internetovým připojením, aplikace nabízí možnost si výsledky vyhledání uložit.

[1] Aktuálně je v offline verzi pouze pro Android a iOS.

IDOS do kapsy plně využívá polohu k nalezení nejlepšího spojení, dokáže dokonce nabídnout i vhodný jízdní řád. Nabízí také širokou škálu možných předvoleb, aby se žádný uživatel nemusel omezovat. Aplikace také podporuje iPad, kde nabízí některé vlastnosti navíc, jako třeba tisk výsledků. [9]

2.7 Srovnání českých vyhledávačů

V tabulce 1 najdeme srovnání výše uvedených českých vyhledávačů vlakových spojení.[2] Srovnáváno je několik základních kritérií, přičemž platí, že je-li aplikace napsána pro více mobilních platforem, důraz je kladen na platformu Windows Phone.

[2]Data jsou aktuální k 12. březnu 2013.

Název aplikace / parametr	WMM Jízdní řády	jVlaky	CGTransit	Pubtran	IDOS do kapsy
Vývojář	Miky WOW	hekrhy	circlegate	F. Hejl	P. Jankuj
Platforma	Windows Phone	Java	Windows Phone, Android, iOS	Android, iOS	iOS
Velikost	2 MB	923 kB	1 MB	880 kB	1.64 MB
Vyhledávání spojení - parametrizace					
Přestupní stanice	NE	ANO	ANO	ANO (nepřesné)	ANO
Volba kategorie spojů	NE	ANO	ANO	ANO	ANO
Vyhledávání podle GPS pozice	ANO	NE	ANO	ANO	ANO
Rozšířené funkce					
Upomínky	ANO	NE	NE	ANO	ANO
Zpoždění vlaku	ANO	ANO	ANO	ANO	ANO
Zobrazení na mapě	ANO	ANO (zjednodušeně)	ANO	ANO (online)	ANO
SMS jízdenka	NE	NE	ANO	ANO (přesměruje do jiné aplikace)	ANO
Práce offline	NE	ANO	NE	NE	NE
Oblíbenost u uživatelů (dle internetového obchodu, kde se aplikace prodává)					
Oblíbenost	4 z 5	-	4,5 z 5	5 z 5	-
Celkové hodnocení					
Cena [Kč]	41,99	zdarma	52,49	zdarma	$2.99
Klady	data z IDOS, upomínky	dobře parametrizovatelná, JAVA, zdarma	jednoduchá aplikace, intuitivní	mnoho nastavení, jednoduchý vzhled	mnoho funkcí podle IDOSu
Zápory	bez offline hledání, občas méně plynulá	pomalá, nepřesné vyhledávání	pro Windows Phone nemá offline vyhledávání	bez offline vyhledávání, občas nekomunikuje s IDOS, reklama	placená aplikace, občas nestabilní

Tabulka 1: Srovnání českých mobilních vyhledávačů spojení

3 Vyhledávání spojů

Vyhledávání spojů je důležitou součástí Asistenční služby pro cestující. V této kapitole se seznámíme s problémemem hledání cesty v grafu a ukážeme si známé algoritmy, ze kterých vychází implementace vyhledávání v asistenční službě.

3.1 Problém vyhledávání

Demonstrujme si nyní problém vyhledávání na železniční síti v Česku. V roce 2013 je zde v provozu celkem 205 tratí [10]. Tyto trati se sbíhají v tzv. *železničních uzlech*. Železniční síť si lze představit jako graf. Nyní se seznámíme s jeho přesnou definicí:

Definice 3.1 *Graf G (také jednoduchý graf nebo obyčejný graf) je uspořádaná dvojice G = (V, E), kde V je neprázdná množina vrcholů a E je množina hran – množina (některých) dvouprvkových podmnožin množiny V . [13]*

Podle této definice se lze snadno dovtípit, že množinu vrcholů V by v našem případě reprezentovaly železniční stanice a zastávky. V E bychom pak nalezli koleje - tedy způsob, jakým jsou jednotlivé stanice propojeny mezi sebou.

Představme si nyní, že chceme najít spojení ze žst. Adamov, ležící v Jihomoravském kraji do žst. Most, která se nachází v Ústeckém kraji. Je zřejmé, že se nejedná o stanice ležící na stejné železniční trati, a tudíž bude třeba projít alespoň jedním *železničním uzlem*, abychom dorazili do cílové stanice. Zajisté však nepřijdeme na nádraží a nezkusíme štěstí tak, že nastoupíme do prvního spoje a budeme doufat, že je to správný spoj. I v reálné situaci totiž využíváme jakéhosi algoritmu podle kterého se řídíme. Abychom si mohli nějaký nadefinovat, řekneme si o našem grafu (neboli železniční síti) něco více.

Obecně platí, že z nějaké stanice A do libovolné jiné stanice B se lze dostat nejméně jedním způsobem. Stejně tak je tomu i obráceně. Jedná se tedy o tzv. *souvislý neorientovaný graf*, což můžeme srovnat s následující definicí:

Definice 3.2 *Graf nazveme* souvislý, *jestliže pro každé dva vrcholy* u,v *je vrchol* v *dosažitelný z vrcholu* u. *To znamená, že existuje taková posloupnost vrcholů a hran (v_0,e_1,v_1,e_2,...,e_n,v_n), kde v_i jsou vrcholy grafu a e_i jeho hrany, přičemž každá hrana e_i má koncové vrcholy $v_(i-1)$ a v_i. [13]*

Pro tyto druhy grafů existuje mnoho grafových algoritmů (např. Dijkstrův, o němž pojednává kapitola 3.2.2). Musíme si však uvědomit, že výše definovaný graf reprezentuje železniční síť. Ta zpravidla není totožná se sítí vlakových spojení. Jinak řečeno, to, že ze stanice A vede železniční trať do stanice B neznamená, že mezi těmito stanicemi jezdí také vlaky.

Zabýváme se tak *grafem s dynamicky se měnícími hranami i vrcholy* (s průběhem dne se různí intenzita provozu a každý spoj může zastavovat v jiných stanicích). Navíc o něm nevíme, zda je v daném okamžiku souvislý (např. pozdě večer jistě nebude možné najít spoj do všech železničních stanic). V sekci 3.2 se seznámíme se základními algoritmy pracujícími s neorientovanými grafy, z nichž vychází algoritmus použitý v aplikaci. O něm pojednává kapitola 5.1.

3.2 Existující algoritmy

Seznamme se nyní s algoritmy, jejichž myšlenky jsou využity v asistenční službě. Jedná se zejména o procházení komponent grafu a jeho modifikaci známou jako Dijkstrův algoritmus. Pro srovnání za nimi následují některé z moderních algoritmů řešících problém vyhledávání.

3.2.1 Procházení souvislých komponent grafu

Pro vyhledávání v dynamicky se měnícím grafu lze s úpravami[3] využít algoritmus *procházení souvislých komponent grafu*. Více se o něm můžete dočíst v [13]. Základní princip činnosti si vysvětlíme na jeho pseudokódu:

```
// na vstupu je graf G
vstup < graf G;
stav(vsechny vrcholy a hrany G) = iniciacni ;
uschovna U = libovolny vrchol u grafu G;
stav(u) = nalezeny;
// zpracovani vybrane komponenty G
while (U je neprazdna) {
    vyber vrchol v a odeber jej z uschovny U: U = U − v;
    ZPRACUJ(v);
    for (hrany e vychazejici z v) { // pro vsechny hrany
        if (stav(e) == iniciacni )
            ZPRACUJ(e);
        w = druhy vrchol hrany e = vw; // zname sousedy?
        if (stav(w) == iniciacni ) {
            stav(w) = nalezeny;
            pridej vrchol w do uschovny: U = U + w;
        }
        stav(e) = zpracovany;
    }
    stav(v) = zpracovany;
    // pripadny prechod na dalsi komponentu G
    if (U je prazdna && G ma dalsi vrcholy)
        uschovna U = {vrchol v_1 z dalsi komponenty G};
}
```

Výpis 1: Algoritmus prohledávání napsaný v pseudokódu

Vidíme, že implementace algoritmu bude jednoduchá a přímočará.
Na začátku

- všem vrcholům i hranám přiřadíme iniciační stav,
- vybereme libovolný vrchol z úschovny.

V průběhu každého cyklu algoritmu zpracujeme nějaký jeden vrchol v. To znamená, že

- zpracovaný vrchol v z úschovny odstraníme,

[3] Tyto úpravy definují zejména stavy potřebné pro prohledání uzlu v různých časech. Více se dozvíme v kapitole věnované dalším algoritmům.

- prozkoumáme (a případně zpracujeme) všechny dosud nezpracované hrany incidentní s vrcholem v,

- pokud je některý vrchol w sousední s vrcholem v v iniciačním stavu, přidáme vrchol w do úschovny U.

Prozkoumáme (a zpracujeme) tak celou komponentu grafu, která obsahuje výchozí vrchol u. Různé implementace tohoto obecného algoritmu lze opět najít v [13].

3.2.2 Dijkstrův algoritmus

Dijkstrův (čti "Dajkstrův"podle nizozemského informatika Edsgera W. Dijkstry) algoritmus je modifikací algoritmu zmíněného v 3.2.1. Tento algoritmus se s úspěchem nasazuje na kladně ohodnocené grafy za účelem vyhledání nejkratší cesty ze zvoleného bodu.

Vysvětleme si nyní myšlenku činnosti Dijkstrova algoritmu. Mějme dán *kladně* vážený graf G s ohodnocením w. Zvolíme jeden vrchol u grafu G za výchozí, z něj budeme hledat vzdálenost do jiného pevně zvoleného vrcholu v grafu G. Graf budeme prohledávat postupem do šířky. Budeme pracovat s následujícími proměnnými:

- počet vrcholů grafu G označíme N,

- předpokládejme, že graf G bude uložen pomocí seznamu sousedů v dvourozměrném poli **sous[][]**, kde prvek **sous[j][k]** udává $(k+1)$. souseda vrcholu j (pozor, protože indexujeme od 0, jedná se o $(k+1)$. souseda, nikoliv o k. souseda),

- pro uložení stupně vrcholů použijeme jednorozměrné pole **deg[]**,

- pracujeme s kladně váženým grafem, váhy budou uloženy v dvourozměrném poli **w[][]**; hodnota MAX_INT bude reprezentovat ∞,

- v průběhu algoritmu budeme pro každý vrchol i ukládat v poli **vzdal[]** informaci o délce nejkratší doposud nalezene cesty do vrcholu i,

- abychom uměli nejkratší nalezenou cestu zrekonstruovat, budeme v poli **pre[]** uchovávat pro každý vrchol i informaci o předposledním vrcholu na nejkratší cestě do vrcholu i,

- protože se jedná o modifikaci algoritmu prohledávání grafu, budeme pro každý vrchol ukládat informaci o jeho stavu do pole **stav[]**; rozlišíme, zda je vrchol v iniciačním nebo zpracovaném stavu.

V každém kroku vybereme z úschovny nalezenych vrcholů vždy ten vrchol j, který má ze všech vrcholů v úschovně nejmenší (dosud nalezenou) vzdálenost od výchozího vrcholu u. Tato informace je uložena v proměnné **vzdal[j]**. [13]

3.2.3 Algoritmus A*

Vyhledávání cesty v grafu pomocí Dijsktrova algoritmu s sebou přináší nevýhodu v podobě expansivního prohledávání celéeho grafu všemi směry. Je zřejmé, že mnoho času ztrácí při prohledávání opačným směrem, než je cílový bod cesty.

Algoritmus A* dodává Dijsktrovu algoritmu jakýsi smysl pro orientaci. Toto provádí pomocí měření vzdušné vzdálenosti aktuálně prohledávaného bodu od cíle. Ukažme si jeho pseudokód:

```
A∗(graf G, ohodnoceni w, vrchol s, vrchol t, heuristika h):
    for each vrchol v V(G):
        d[v] = ∞, pre[v] = NULL;
        S = {}, d[s] = 0;
        while |V(G)−S| > 0:
            u = min_{d+h}(V(G)−S);
            if (u == t):
                break;
            S = S ∩ {u};
            for each hrana (u,v) ∈ E(G):
                if (d[u] + w(u,v) < d[v]):
                    d[v] = d[u] + w(u,v);
                    pre[v] = u;
    return RekonstrukceCesty(pre,t);
```

Výpis 2: Algoritmus A* (pseudokód)

Algoritmus v ideálním případě přináší zrychlení prohledávání, jelikož již volbou správného směru při hledání z první stanice může zredukovat počet procházení až na polovinu. Ve speciálních případech však může selhávat. Stává se tak v tehdy, je-li do cílové stanice nutno jet dlouhou úvratí (neboli po nějakou dobu se vzdálenost do cíle zvyšuje). Výhody algoritmu se pak zmenšují tím více, čím dříve dojde k chybnému odhadu směru.[11] Tuto nevýhodu lze částečně odstranit zavedením grafu železniční sítě.

3.2.4 Reach-based vyhledávání

Tato technika moderního vyhledávání je založena na extensivním předzpracování dat. Každý vrchol v grafu je součástí omezeného množství nejkratších cest. Na každé takové cestě vrchol v propojuje dva nezávislé úseky. Menší z obou přímých vzdáleností ke koncim úseků je parametr, který určuje význam samotného vrcholu v na dané cestě. Lze tedy zjistit nejvyšší význam mezi všemi nejkratšími cestami, které daným vrcholem prochází. Pokud se nám podaří předem zajistit pro každý vrchol tuto statistiku, můžeme během vyhledávání nejkratších cest některé vrcholy rychle vyloučit. Pokud hledáme cestu do vzdáleného vrcholu, je zbytečné prohledávat na půli cesty vrcholy s nízkým předpočítaným významem.

Postup tedy slouží jako přesná heuristika pro další algoritmy a lze ho vhodně kombinovat s dalšími technikami. Potíží však zůstává fáze rozsáhlého preprocessingu dat a přizpůsobení se dynamickým změnám v datech.

Všimněme si faktu, že méně důležité prvky v síti se prohledávají pouze v blízkosti počátku a cíle hledané cesty. To je základní myšlenka využívaná přímo i nepřímo v řadě dalších algoritmů.[11]

3.2.5 Algoritmus v Asistenční službě

Algoritmus implementovaný v Asistenční službě pro cestující vychází z výše uvedených algoritmů procházení souvislých komponent grafu (viz kap. 3.2.1) a také Dijsktrova algoritmu (kap. 3.2.2). Jeho návrh a popis naleznete v kapitole 5.1.

4 Návrh aplikace asistenční služby

4.1 Motivace a výhody použití asistenční služby

Asistenční služba pro cestující je komplexní aplikace zaměřená zejména na pohodlné vyhledávání (nejen) vlakových spojů. Hlavní motivací tohoto projektu bylo poskytnout cestujícímu pomocnou ruku při cestách zejména po neznámých krajinách. Velmi často je totiž odkázán na pomoc druhých, cestuje-li do neznámých míst. Stále komplikovanější systém přestupů, vzrůstající počet kategorií spojů a dopravců a mnohdy i nedostatek informací způsobují chaos, což může mít za následek i špatnou volbu spoje.

Aplikace si dává za úkol nabídnout všechny potřebné informace o spoji, o blížící se stanici i o nástupišti a koleji, odkud náš spoj pojede[4]. V ideálním případě má cestující možnost hledat alternativní způsob dopravy do cíle, má-li jeho spoj zpoždění a existuje riziko[5], že daný spoj nestihne.

4.2 Uživatelé asistenční služby

Asistenční služba pro cestující zavádí tři typy uživatelů - *administrátor*, *registrovaný cestující* a *cestující*.

4.2.1 Administrátor

Administrátor je uživatel, který má v aplikaci zavedeno administrátorské uživatelské jméno a heslo, jejichž prostřednictvím se přihlašuje do *webové* části aplikace. Je nejvyšší autoritou systému.

Po přihlášení do systému může spravovat data jízdních řádů jako je seznam spojů, jejich řazení či detail jízdy (stanice a zastávky). Administrátor rovněž rozhoduje, kdy a jakým způsobem bude vydána aktualizace.

4.2.2 Registrovaný uživatel

Stojí v pomyslné hierarchii pod *administrátorem*. Je řádně zaregistrován ve webové části aplikace a smí přidávat nové *body zájmu*. Tyto body se v mobilním klientovi zobrazí až tehdy, jsou-li schváleny některým administrátorem.

4.2.3 Cestující

Cestující je každý uživatel *mobilního* klienta. Smí využívat veškerá data, která byla dodána s aplikací včetně všech vydaných aktualizací, nemůže je však editovat a přidávat. Cestující může být zároveň i *registrovaným uživatelem*.

[4] Pro zjištění informací o návazném spoji je potřeba být připojen k internetu.

[5] Jak aplikace zjišťuje riziko rozvázání přípojné vazby se dozvíte níže.

4.2.4 Neregistrovaný uživatel

Nejedná se o uživatele v pravém slova smyslu. Je to každý návštěvník webového klienta, který nemá přihlašovací jméno a heslo. Neregistrovaný uživatel nemá právo na úpravu dat, smí je pouze prohlížet.

4.3 Struktura projektu

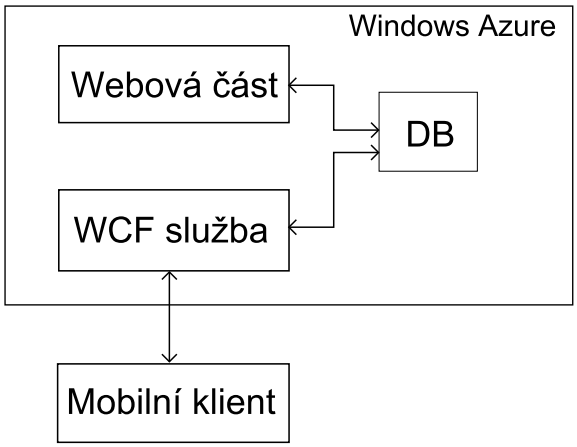
Projekt je rozdělen do 3 základních celků:

- *webová část* – aplikace běžící na cloudovém serveru, implementovaná v ASP.NET. Zde se provádí veškerá administrativní činnost, úpravy jízdních řádů, práce s body zájmů apod. Odtud jsou také spravovány aktualizace.

- *mobilní klient* – klient napsaný pro platformy Windows Phone 7 a 8. Jeho funkcionalita je rozdělena na dvě skupiny, podle toho, zda využívá internetovou konektivitu či nikoliv. Při návrhu i implementaci byl kladen důraz na to, aby co nejvíce funkcí fungovalo autonomně a bez potřeby připojení k internetu.

- *WCF služba* - služba zastřešující komunikaci mezi mobilní a webovou částí, může rovněž fungovat jako agregátor informací o spojích (např. poloha vlaku, jeho zpoždění apod.)

Jednotlivé části reprezentují skupiny funkcí splňující výše uvedené požadavky. Komunikaci mezi jednotlivými celky znázorňuje obrázek 1. Z něj je patrné, že veškerá výměna informací mezi mobilním klientem (či webovou částí) a *SQL Azure* databází probíhá pouze prostřednictvím WCF služby.



Obrázek 1: Schéma komunikace mezi jednotlivými vrstvami systému

WCF služba rovněž může poskytovat data o zpoždění vlaku a přípojích[6] bez nutnosti připojení k databázi. Data o poloze vlaku mohou být získávána například ze známého serveru `http://babitron.ic.cz`.

4.4 Návrh databáze

Webový klient i WCF služba komunikují s *SQL Azure* databází, která uchovává veškerá data o jízdních řádech a bodech zájmu. Návrh databáze je patrný ze schématu na obrázku 2. Popíšeme si nyní nejzajímavější tabulky a jejich význam.

Jedním z požadavků při návrhu databáze byla inteligentní propagace aktualizací. Jak bylo zmíněno v sekci 4.2, administrátor rozhoduje o vydání aktualizace. Jakmile se tak stane, veškerá neaktuální data pozbývají platnosti a měla by být odstraněna. V tabulkách, podléhajících možným aktualizacím z webové části, tak nalezneme atribut *TimeStamp*. Pomocí tohoto časového razítka lze kontrolovat, zda byla data od poslední aktualizace, kterou máme v mobilním klientovi, přidána, či změněna. To, že byla data smazána, detekujeme pomocí atributu Deleted (některá data nejsou na webové části mazána, ale pouze zneplatněna). Souhrnnou informace o aktualizacích najdeme v tabulce *UpdateLog*.

Informace o databázi jsou uloženy v podobě dvojice klíč, hodnota v tabulce *DatabaseInfo*. Ta obsahuje zejména údaje o platnosti jízdního řádu, který využíváme v mobilním klientovi, o jeho názvu a další pomocné informace.

Čtenáře bych chtěl také upozornit na tabulku *TrainRestriction*. V *Restrictions* této tabulky jsou uložena omezení v jízdě pro dané spoje. Najdeme zde 403 znaků[7] 1 nebo 0. Jednička symbolizuje, že spoj v daném dni jede, nula že nikoliv. Platí, že pro určení dostupnosti spoje je rozhodující den *opuštění výchozí stanice vlaku*. Tímto jsou ošetřeny problémy například u dálkových nočních spojů jedoucích přes půlnoc.

Součástí mobilního klienta jsou funkce usnadňující přestup v jednotlivých stanicích. Abychom získali informace o přípojích, potřebujeme mít přístup k informačním tabulím železničních stanic. Tabule najdeme na adrese `http://provoz.szdc.cz/Tabule`, kde každá stanice má své, pevně vygenerované, identifikační číslo. Tato čísla ukládáme v tabulce *StationWebParameter*.

Zejména v beta verzi programu vývojář ocení rychlou zpětnou vazbu v případě selhání. V tabulce *ExceptionLog* najdeme chybové zprávy uložené pomocí WCF služby.

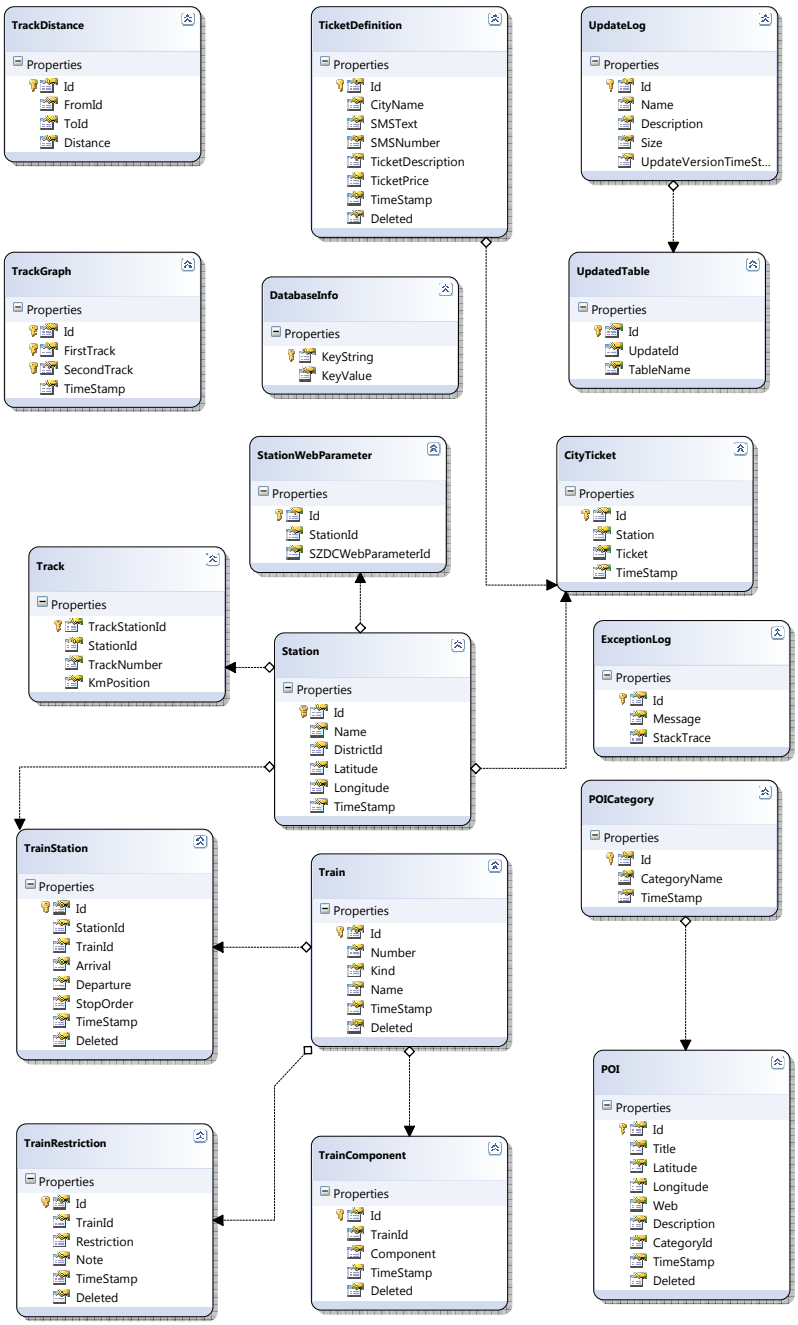
4.5 Získávání a zpracovávání dat

V předchozím textu se čtenář mohl seznámit s návrhem schématu databáze na SQL Azure. Nyní si ukážeme, odkud a jakým způsobem byla získána data. [8]

[6] K využívání těchto dat je třeba souhlasu majitele. V asistenční službě jsou proto získávána fiktivní data vygenerovaná přímo službou.

[7] Pro zjednodušení vyhledávání je pro každý měsíc v roce vyhrazeno 31 znaků, přičemž nevyužité znaky jsou označeny libovolným znakem. Začíná se prosincem předchozího roku a končí prosincem tohoto, proto je ukládáno 13 měsíců

[8] Získávání jakýchkoliv dat z jízdních řádů je vázáno smluvními podmínkami vlastníka. Bohužel se mi nepodařilo od zúčastněných stran získat potřebný souhlas, a proto zde uvádím způsob, jakým by bylo možné data získat. V aplikaci Asistenční služba pro cestující jsou k dispozici jen data ze starých jízdních řádů.



Obrázek 2: Schéma databáze

Nejdůležitějšími daty jsou informace o jízdě spojů (tedy příjezd a odjezd z/do dané stanice). Nejjednodušší cestou bylo je získat z nějaké webové aplikace, kde již byla zpracována do tisknutelné podoby, aby bylo jednodušší je vyparsovat. Nejvhodnější z nich je dostupná na webové adrese **http://blind.jizdnirady.idnes.cz**. Zde jsou jednotlivé spoje uloženy s minimem grafických prvků v tabulkách, a tak je velmi jednoduché vytvořit parser, který data získá.

Pro každý spoj je k dispozici vygenerována samostatná stránka, jejíž adresa vypadá například takto: **http://blind.jizdnirady.idnes.cz/Route.asp?i=176539**, kde parametr *i* je id spoje. Aplikace tedy prochází jednotlivá id a získává obsah stránky. Na serveru jsou uloženy spoje jedoucí po celé Evropě, ale do asistenční služby potřebujeme získat jen ty, které jedou po území ČR. Tohoto dosáhneme velmi jednoduše, jelikož vlaky jedoucí po našem území jsou uváděny vždy jako první. Parseru tak stačí pomocí hlavičky stránky zkontrolovat, zda spoj s daným číslem již nebyl dříve uložen [9] a zda má jeden ze známých typů. [10] Data jsou pak pouze převedena a uložena do databáze.

Stažení a zpracování velkého množství dat je časově náročná činnost. Navíc se projevuje i latence při připojení. Z těchto důvodů je parser rozdělen do několika vláken (stahovací, několik zpracovávacích a ukládací), Takto jsou data získána a zpracována v rozumném čase. Činnost jednotlivých vláken odpovídá jejich názvům - stahovací vlákna ukládají kompletní HTML kód stránky do bufferu, odkud jej vybírají zpracovávací. Tyto po rozprašování přenechají výsledek operace ukládacímu vláknu, které se postará o správné uložení do databáze.

Po získání dat o jízdních řádech spojů bylo nutno získat řazení vlaku. To je velmi dobře zpracováno na serveru `www.zelpage.cz`. Schéma adresy, ze které lze řazení získat, je následující:

http://www.zelpage.cz/razeni/*rok*/vlaky/*dopravce-cislovlaku*?zobrazeni=text
Kde:

- **rok** jsou poslední dvě číslice roku platnosti jízdního řádu (např. 13)
- **dopravce** je označení dopravce spoje (cd, rj nebo le)
- **cislovlaku** je jedinečné číslo spoje (např. 1485)

Poslední parametr pak říká, že data mají být zobrazena v textové podobě. Z webu jsou získávána pouze kódová označení lokomotiv a vozů, které jsou pak dále zpracovávána až při zobrazení v mobilním klientovi.

Zpracování informací o omezení jízdy spojů je o něco složitější. Data lze nalézt v textové podobě na `http://www.iwan.eu07.pl/jw/2013/`, kde jsou k dispozici pomůcky k aktuálnímu grafikonu vlakové dopravy. Zpracování textu pro potřeby mobilní aplikace je založeno na analýze pravidel, jejichž pomocí jsou tvořeny poznámky o jízdě vlaků. Ačkoliv tato analýza není hlavní náplní této práce, lehce si nastíníme, jakým způsobem probíhá.

[9] Tento způsob kontroly v několika málo případech selhává, jelikož čísla spojů v ČR nejsou jedinečná. Naštěstí, těchto případů není mnoho, a tak lze chybějící vlaky bez problémů doplnit.

[10] Jedná se o typy spojů jedoucích po území ČR. Takto lze odstranit spoje s těmi čísly, které u nás nejezdí.

Text poznámky vždy začíná slovy *jede [v]*, případně *nejede [v]* následovanými několika daty či čísly označující dny v týdnu[11] a případně opět slovy jede či nejede.

Tato data mohou být uvedena buď jednotlivě, a nebo rozsahově (např. 25. - 28.XII). Pro snadnější zpracování algoritmem byla rozsahová data převedena do výčtu jednotlivých dní. Každému spoji je přiřazena booleovská matice o 13 řádcích a 31 sloupcích. Při zápisu do matice je využíváno logických operací *and* a *or*.

Algoritmus řešení postupuje takto:

- Vstupem je matice booleovských hodnot, přidělující každému dni příznak, zda v daný den spoj jede, či nikoliv.
- Začíná-li text slovy *jede* (případně jede v), znamená to, že následující text bude označovat dny, kdy spoj jede. Ve dnech nepatřících do tohoto seznamu pak spoj nepojede. Algoritmus tedy v tomto případě přednastaví všechny prvky matice na **false**
- Začíná-li text slovy *nejede* (případně nejede v), prvky matice budou nastaveny na **true**.
- Konkrétní dny a dny v týdnu jsou jednotlivě zaznamenávány do matice opačnou hodnotou, než byla přednastavena
- Následuje-li opět text jede nebo nejede, algoritmus pokračuje ve zpracovávání s tím, že do matice zaznamenává opačnou hodnotu.
- Matice je převedena do řetězce o 403 znacích a uložena do databáze

Poznámka: Text v některých případech obsahuje záznamy typu *jede 24./25.XII.*. To znamená, že daný spoj jede tzv. přes půlnoc (například odjíždí z výchozí stanice 24.XII., do cíle však dorazí následujícího dne). Princip omezení jízdy vlaku v mobilním klientovi zohledňuje pouze den odjezdu z výchozí stanice, a tak lze text upravit na *jede 24.XII.* beze změny významu této informace.

[11] Popř. také + a X coby dny pracovního klidu a pracovní dny

5 Mobilní klient

Mobilní klient je napsán pro operační systém Windows Phone 7.5 (Mango). Po příchodu nového systému Windows Phone 8 byl klient upraven, aby fungoval také na této platformě. Aplikace je napsána v Silverlightu a XAMLu. Cílem této mobilní aplikace je poskytnutí komplexních služeb pro cestujícího s důrazem na rychlost a minimalizaci závislosti na internetovém připojení. Rozdělení aplikace je následující:

- vyhledávač vlakových spojení
- průvodce cestou
- prohlížeč dat
- manažer aktualizací
- zpětná vazba při chybových stavech

Pro využití některých funkcí je nutná komunikace s WCF službou. Tato služba běžící v cloudu poskytuje informace o aktuální pozici vlaku a přípojích.

5.1 Vyhledávač vlakových spojení

Stěžejní částí mobilního klienta je parametrizovatelný vyhledávač vlakových spojení. Kromě základních věcí, jako je zadání odkud, kam a kdy chce cestovat, má uživatel možnost definovat také typ vlaku. Cestující může nastavit rovněž počet přestupů a celkovou dobu jízdy.

Výchozí stanici je možno také vybrat ze seznamu nejbližších stanic od naší pozice. Aplikace také umožňuje volbu cílové stanice pomocí volby z tzv. bodů zájmu[12]. V takovém případě sama vyhledá nejbližší vlakovou stanici k tomuto bodu a nabídne spojení.

Pro potřeby aplikace jsem se rozhodl sestavit algoritmus, který vychází z procházení souvislých komponent grafu a Dijkstrova algoritmu, o nichž jsem se zmínil výše. Vyhledávání probíhá na mobilním zařízení, a tak bylo důležité, aby pro svou činnost potřebovalo pouze minimální množství dat.[13] Také z tohoto důvodu algoritmus nepracuje s kompletní znalostí trati, ale vždy jen s informacemi o lokálním uzlu(vrcholu) a vrcholech k němu přímo připojeným.

V průběhu algoritmu se provádí odstranění spojů, které ten den nejedou a také těch, jejichž typ uživatel vyřadil před samotným hledáním. Více o této činnosti se dozvíte v kapitole 5.1.2.

Popišme si nyní jednotlivé kroky algoritmu v analogii s vlakovou dopravou:

- Všechny vrcholy (stanice) jsou nenavštíveny. Vložíme výchozí stanici ve zvoleném čase do fronty .
- Dokud není fronta prázdná, opakujeme následující kroky:

[12] Funkce je dostupná po stažení volitelné aktualizace POI bodů

[13] Důvod byl mj. v nutnosti držet data v paměti telefonu, jak bude zmíněno dále v textu.

  - Vezmeme první prvek z fronty
  - Zjistíme všechny ten den jedoucí spoje povoleného typu[14] odtud odjíždějící do určité doby [15].
  - Je-li tento vrchol přímo propojen s cílovým vrcholem, ukončíme hledání. **Nalezli jsme úspěšně spojení**.
  - V opačném případě
    * Vložíme do fronty všechny nácestné stanice (tj. stanice, jenž nás ještě "čekají") všech spojů společně s časem příjezdu do dané stanice, pokud zde již nebyly.
    * Byla-li některá ze stanic již ve frontě v horším čase, než dosud, odstraníme ji z fronty a zařadíme do fronty aktuální situaci
    * Byla-li některá ze stanic již ve frontě v lepším čase, aktuální situaci do fronty nevkládáme

- Je-li nyní fronta prázdná a nebyl nahlášen úspěch, ohlásíme neúspěšné hledání.

Pro nalezení optimálního řešení je nutno vyprázdnit frontu potenciálních lepších výsledků. Toto je značně neefektivní vzhledem k tomu, že fronta obsahuje velké množství tzv. slepých uzlů (tj. uzlů, které nemají potenciál, abychom skrze ně nalezli lepší řešení, než dosud nalezené). Navíc platí, že čím větší vzdálenost mezi oběma koncovými body bude, tím více takovýchto uzlů bude ve frontě. V části 5.1.1 uvádím postupně prováděné optimalizace vyhledávání.

Při počítání lepší varianty v daném uzlu nelze ukládat čas příjezdu. Důvod se pokusím demonstrovat na následující situaci: Do stanice Ostrava-Svinov jsme dosud nalezli nejlepší čas příjezdu ve 23:58. Nalezneme-li nyní příjezd v 2:02, algoritmus by jej prohlásil za lepší, jelikož 23:58 $>$ 2:02, ačkoliv ve skutečnosti je zde tento spoj o více než 2 hodiny později!

Jakmile máme nalezeno cíl, je třeba zrekonstruovat celou cestu a zobrazit ji uživateli. V průběhu vyhledávání si aplikace vytvořila několik stromů. Kořenem každého z nich je výchozí stanice hledání a *unikátní* čas odjezdu spoje z ní. Uzly jsou pak tvořeny přestupními stanicemi včetně informací o čase přestupu a spoji, ze kterého přestupujeme. Právě jeden ze stromů obsahuje v listu cílovou stanici a spoj, kterým se do ní dopravíme. Při rekonstrukci cesty algoritmus plně využívá vlastností stromu a pracuje takto:

- Vezme spoj, kterým bychom dorazili do cíle a označí jej jako **aktuální**
- Dále v cyklu
  - Projde *v obráceném pořadí* stanice **aktuálního spoje** a ukládá je do zásobníku
  - Narazí-li na stanici uvedenou v rodičovském uzlu, označí zde uvedený spoj jako **aktuální**

[14] Více v kapitole 5.1.2

[15] Tuto dobu lze měnit v nastavení jako *dobu přestupu*

  - Pokud je aktuálně uložená stanice výchozí pro naše hledání, ukončí cyklus

- Projde zásobník a všechny záznamy v něm doplníme o čísla spojů, názvy stanic atp.

5.1.1 Implementační detaily

V textu výše jsem nastínil některé z důvodů, proč byl navržený algoritmus poměrně pomalý. Jistě je každému čtenáři jasné, že operace načítání každého spoje z databáze je velmi zdlouhavá[16], a proto pracujeme s daty uloženými do operační paměti zařízení. Windows Phone nám k tomu poskytuje pole objektů s názvem *state*.

Nicméně, tato úprava, byť přináší razantní zlepšení, není dostačující. Je jistě zbytečné vyhledávat v uzlech v čase, ve kterém již máme dosaženo lepšího výsledku v cílové stanici. Narazí-li tedy algoritmus na daný prvek fronty, ani jej nezkoumá a rovnou jej z fronty vyhodí. Tato část plně vychází z myšlenek Dijkstrova algoritmu. Stejně tak se chová i v případě, že jsme v dané stanici větším množstvím přestupů, než si uživatel nastavil.

Je také třeba zmínit, že aplikace v průběhu hledání pracuje pouze s pro vyhledávání nezbytně nutnými daty. Rozeberme si nyní, která data o stanicích a spojích potřebujeme. V předchozí kapitole jsem se zmínil o tom, že není nutné znát v každém okamžiku celou trať, ale pouze lokální informaci. Podíváme-li se na algoritmus hledání, zjistíme, že často pracuje pouze buď s příjezdem nebo s odjezdem vlaku (avšak ne s oběma naráz). Je zřejmé, proč tomu tak je – hledáme-li přestup, zajímá nás totiž pouze to, v kolik hodin v této stanici budeme a v kolik hodin odjíždí přípojný vlak.

Jednou z nejnákladnějších operací při vyhledávání spojení je třídění podle času příjezdu či odjezdu. Při pohledu na algoritmus je navíc zřejmé, že k této operaci by muselo docházet velmi často. Jistě je však zbytečné je setřizovat pokaždé. Stačí nám, abychom měli data setřízena už v paměti a načítali je sekvenčně. Narazíme však ještě na jeden problém - třídění většího množství dat [17] na mobilním zařízení je pomalá operace. Z tohoto důvodu budeme setřízená data přímo ukládat do databáze, čímž využijeme vlastnosti tabulky typu *halda*, tedy toho, že uspořádání dat je dáno čistě pořadím, v jakém byly prvky vloženy. V tabulce totiž nebudeme mazat[18] a ani editovat.

5.1.2 Parametrizace vyhledávání

Již dříve v textu jsem se zmiňoval o omezení jízdy spojů, tedy období, kdy spoj nejede vůbec, nebo jen v nějakém konkrétním úseku. V takovém případě je tento spoj uložen do databáze v každé své variantě zvlášť.

Připomenu, že v databázi mobilního klienta je pro tyto účely uložena tabulka s názvem *TrainRestriction*. Zde, ve sloupci *Restrictions*, najdeme řetězec 403 znaků 0 nebo 1, podle toho, zda spoj v daném dni jede. Při vyhledávání odjezdů z železniční stanice se

[16] Exaktně se o tom přesvědčíme v části věnované testování aplikace
[17] Jízdní řád má cca 93 000 příjezdů a odjezdů.
[18] Pouze zneplatňovat

pak program jednoduše „podívá“, zda znak na příslušné pozici je 1, pokud ne, odstraní spoj z vyhledávání. Pro redukci počtu záznamů v tabulce jsou zde ukládány jen spoje, které nejedou denně.

Poznámka: Zdálo by se, že hromadné odstranění spojů, které ten den nejedou, před samotným vyhledáváním, by bylo efektivnější. Je třeba brát v potaz, že hledání může „přesahovat“ půlnoc. V takových případech by tento naznačený postup vedl k neplatným výsledkům hledání, jelikož bychom mohli odstranit nesprávné spoje.

5.2 Průvodce cestou

Průvodce cestou nabízí přístup k rozšířeným funkcím asistenční služby, jako jsou určení polohy, zpoždění vlaku, zjišťování přípojných vazeb apod. V této části práce si jednotlivé funkce popíšeme.

5.2.1 Moje cesty

Jakmile cestující vyhledá spojení, může si jej uložit do tzv. mých cest. Moje cesty poskytují jednak itinerář jízdy, ale také přístup k mnoha dalším funkcím dostupným během jízdy (např. určení polohy našeho či přípojného vlaku). Při ukládání je k dispozici několik parametrů, jako je povolení přístupu k internetu atp.

5.2.2 Určení polohy spoje

Častou komplikací při cestování, zejména pak máme-li přestupovat či vystoupit v cíli naší cesty, je neznalost železničních stanic. Cestující jsou často odkázáni na pomoc spolucestujících či vlakového personálu. Tato komplikace ještě vzrůstá, jsme-li při přestupování v časovém presu. V sekci 5.4.2 se můžete dočíst, jaké nástroje aplikace pro usnadnění přestupu nabízí.

Představme si nyní modelovou situaci. Sedíme ve vlaku, který projíždí nám naprosto neznámým terénem. Rádi bychom šli do restauračního vozu, ale protože má vlak zpoždění (jehož velikost neznáme), nejsme si jistí, zda bychom se stihli vrátit včas. Jiný cestující si doma na papír poznačil název přestupní stanice. Bohužel však netuší, kolik zastávek a jak daleko od této stanice je.

Oběma těmto cestujícím jistě přijde vhod možnost určení polohy vlaku. Pomocí GPS aplikace najde aktuální pozici a oznámí jim, kde se nachází. Je zbytečné se nyní zmiňovat o způsobu zjištění této polohy, jelikož se o to ve všech moderních operačních systémech postará nějaké API a v aplikaci jen zpracujeme výsledky.

V původním návrhu aplikace jsem zamýšlel určovat polohu pouze z jedné získané GPS pozice. Cílem tohoto omezení byl požadavek[19], abychom získali přesnou pozici i v případě, že náš vlak zrovna stojí např. u vjezdového návěstidla.

Jak již bylo zmíněno výše, železniční síť je grafem. Pomocí jedné GPS pozice jsme schopni určit bod, kde se nacházíme. Tento bod pak lze pomyslně brát jakou součást hrany mezi dvěma vrcholy grafu - tedy mezi železničními stanicemi. Nám však nestačí

[19] Tento požadavek vznikl na základě vlastních potřeb.

vědět, mezi kterými stanicemi se náš vlak nachází[20], potřebujeme totiž zjistit, ke které stanici se blížíme.

Aby nám aplikace mohla poskytnout tuto informaci, musela by „znát“ schéma železniční trati, po které náš spoj jede. To by přineslo buď velký nárůst velikosti aplikace (v případě uložení offline), a nebo závislost na internetovém připojení, což bylo neakceptovatelné.

Nadějnou možností bylo využít znalosti pořadí železničních stanic, v jakém má jimi vlak projíždět. Tato varianta funguje bez problémů na většině tratí. Selže však v případě, kdy vlak na své cestě změní směr jízdy (ať už jízdou po oblouku, který vlak „otočí“ nebo přepřahá-li lokomotivu, jak tomu je například v Brně či Přerově). V takovéto situaci pak nemusí platit, že nejbližší nácestná stanice je skutečně stanicí, která má nyní následovat. Naopak bychom cestujícího snadno zmátli, že je mnohem dál, než ve skutečnosti je.

V aplikaci bychom mohli využít funkce *Course*, která nám vrací heading vzhledem k tzv. pravému severu. Díky tomu by bylo možné pouze z jediné získané pozice určit naší aktuální polohu, avšak správný výsledek by byl příliš závislý na tom, zda sedíme po či proti směru jízdy a také na přesnosti funkce, na kterou se nelze příliš spolehnout. [21]

Z důvodů zmíněných v předchozím textu bylo zřejmé, že je třeba použít většího počtu měření. V extrémním případě, kdybychom prováděli zjišťování polohy vlaku nepřetržitě, bylo by zjištění naší polohy snadné. Bohužel by nám již při středně dlouhé cestě nestačila kapacita baterie.

Aplikace tedy volí jakýsi kompromis a používá pro určení polohy vlaku vůči železniční stanici dvou na sobě nezávislých měření. Abychom eliminovali výše nastíněný problém při změně směru jízdy, kombinujeme tato měření se znalostí pořadí železničních stanic. Získali jsme tak zjišťování polohy, které sice funguje jen v případě, že je vlak v pohybu, na druhou stranu však nespotřebovává tolik energie z baterie.

5.2.3 Upozornění na zajímavá místa podél trati

Cestování vlakem na delší vzdálenosti bývá velmi vyčerpávající a často i nudné. Jedeme-li však neznámou krajinou, mnohdy si zpříjemníme cestování sledováním okolní krajiny. Vidíme tak mnoho památek, které jsme málokdy schopni správně určit.

Jednou z volitelných aktualizací mobilního klienta je databáze bodů zájmu tzv. POI bodů. Tyto POI body obsahují informace o různých zajímavých místech, které lze vidět z vlaku. Cestující tak snadno zjistí, o jakou památku se jedná. Navíc, jsou-li k dispozici, může zjistit další informace či případné recenze této památky.

Pro asistenční službu byly body zájmu získány zejména z volně stažitelných GPX souborů. Autorem bodu zájmu se může stát jakýkoliv uživatel, který má platný účet ve webové části (můžeme si připomenout role uživatelů zmíněné v 4.2).

[20] V dalším textu navíc zjistíme, že nám jedno měření nepostačí ani k tomuto účelu.

[21] Při testování jsem občas získal neplatné údaje.

5.2.4 Nákup jízdenky na MHD

Bylo-li cílem naší cesty nějaké větší město, velmi pravděpodobně budeme dále pokračovat městskou hromadnou dopravou. Asistenční služba k tomuto účelu poskytuje možnost zakoupení SMS jízdenky. Jsme-li ve městě, kde je tato služba poskytována [22], nabídne nám možnost výběru příslušné jízdenky. Po jejím zvolení předvyplní text SMS zprávy a číslo příjemce, takže nám stačí ji pouze odeslat.[23]. Nejpozději za pár minut nám přijde SMS jízdenka a my můžeme pohodlně cestovat.

Princip této funkce je velmi jednoduchý. Mobilní klient má v databázi uložen seznam železničních stanic příslušících k dané MHD. Jakmile z GPS zjistíme, že nejbližší stanice patří k nějaké MHD, zobrazíme ceny jízdenek v této lokalitě. Zjišťování tímto způsobem není nejpřesnější metodou, jelikož nejbližší stanice může být vzdálena i 10 km, avšak vzhledem k minimálnímu nárůstu dat nabízí rozumnou míru přesnosti.

Poznámka: Mohlo by se zdát, že je mnohem jednodušší volit stanice podle názvu - zde však narazíme na to, že existují stanice, které neobsahují ve svém názvu město, do nějž přísluší (např. *Polanka nad Odrou* je součástí MHD Ostrava). Nelze jít ani cestou měření vzdálenosti od hranic města. Museli bychom totiž navíc ukládat tyto hranice a jejich zpracování by vzhledem k nepodporovaným spatial objektům v databázi (viz např. [14]) by bylo výpočetně náročné.

5.3 Další funkce dostupné offline

Asistenční služba pro cestující nabízí mnoho funkcí, jejichž data jsou uložena přímo v mobilním klientovi, a tak není nutno využívat internetového připojení.

5.3.1 Seznam stanic a jejich spojů

Občas si potřebujeme prohlédnout seznam příjezdů či odjezdů vlaku ze zvolené železniční stanice. Asistenční služba pro cestující má tato data uložena v lokální databázi. Získávat je však odtud by bylo značně pomalé a velmi by snižovalo uživatelský komfort.

Z paměti jsou data získávána pomocí slovníku stavu. Tento slovník je uložen v **PhoneApplicationService.Current.State**. Kód k tomu určený je jednoduchý:

```
IDictionary<string, object> state = PhoneApplicationService.Current.State;
Dictionary<int, StationLDC> stations = (Dictionary<int, StationLDC>)state["stations"];
```

Výpis 3: Získávání dat přímo z paměti

Tento kratičký kód je v různých podobách používán při každém vyhledávání spojení a také zobrazování stanic a tratí.

5.3.2 Vlaky, detail trasy a řazení

Uživatel má rovněž k dispozici seznam všech spojů v databázi. Ke každému vlaku získáme informace o jeho kompletním jízdním řádu, omezení provozu a také řazení vlaku.

[22] K 19. březnu 2013 tomu tak bylo v Praze, Ostravě, Olomouci a Ústí nad Labem

[23] Pro správné fungování služby je nezbytné mít povoleny tzv. Premium SMS

To je nabízeno dvěma způsoby: Pro laika jsou zobrazeny jen základní informace (tj. jedná-li se o lokomotivu, vůz 1., 2. třídy či restaurační vůz)[24]. Tyto údaje doplňuje přesný typ daného vozu. Zkušenější uživatel se tak dozví více informací.

5.4 Funkce využívající internet

Funkce využívající internet jsou poskytovány pomocí WCF služby prostřednictvím tzv. SOAP zpráv. Tato služba je umístěna v cloudu. Pro komunikaci s mobilním zařízením je k dispozici pouze asynchronní způsob komunikace. Lze přenášet pouze jeden, předem zvolený, typ kolekce (v našem případě „klasické“ pole) a jeden typ slovníku.

Omezení najdeme i v bezpečnosti komunikace, jelikož lze použít pouze základní typ vazby bez zabezpečení. Šifrování je tak čistě na obou komunikujících stranách.

Mobilní klient obsahuje dva způsoby ošetřování případných výjimek:

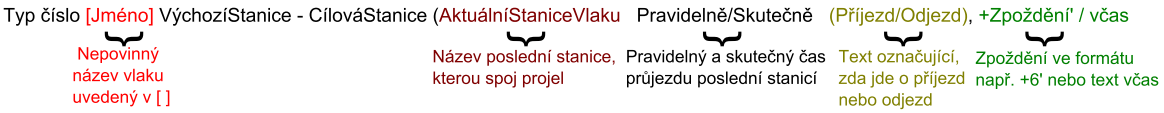
- *Kontrola, zda nedošlo k výjimce při zpracování* - provádí se pomocí property **Error** vyvolané event handlerem metody označené názvem operace WCF služby a suffixem Completed (např. GetAllTrainsToStorageCompleted)
- *Odchycení výjimek EndpointNotFoundException a CommunicationException* v téže metodě - tyto výjimky jsou zpravidla vyvolány tehdy, nastane-li problém přímo při komunikaci se službou (např. v případě selhání internetového připojení či samotné služby).

Poznámka: Výjimka CommunicationException je ekvivalentem k EndpointNotFoundException v operačním systému Windows Phone 8, se kterým je tak aplikace také kompatibilní.

5.4.1 Informace o poloze vlaku

Zpoždění vlaku je poměrně častým a nepříjemným jevem, se kterým se jistě každý, kdo cestuje, potkal. Aby se cestující mohl na tuto situaci předem připravit, aplikace poskytuje informace o poloze vlaku.

Tato data jsou získávána WCF službou ze serveru `http://babitron.ic.cz`. Tento server jsem si vybral proto, že poskytuje data ve velmi snadno parsovatelné podobě. Každý spoj je zapsán na jednom řádku v podobě uvedené na obrázku 3.



Obrázek 3: Struktura zápisu o zpoždění na babitron.ic.cz

Zápis tak může vypadat například takto:
Ex 146 [Kysuca] Žilina – Praha hl.n.: Mosty u Jabl. 12:59/13:09 (odj.), +10′

[24] Jsou vygenerovány dle známých pravidel o značení spojů v ČR.

Na serveru `http://babitron.ic.cz` jsou k dispozici tři stránky s těmito údaji. Jedná se o rychlíky, 4-ciferné spěšné a osobní vlaky a 5-ciferné osobní vlaky. Před vyhledáním příslušného spoje je třeba vybrat správnou stránku. V případě, že vyhledáváme 4-ciferný spoj, je třeba projít první dvě jmenované (jelikož existují také 4-ciferné rychlíky, byť jich není mnoho). Protože rychlíků je mnohem méně, než osobních vlaků, procházíme je jako první a v případě úspěchu už dále nehledáme.

Při parsování bylo třeba dbát na to, že název vlaku není povinný. Stejně tak nemusí být informace o spoji aplikaci Babitron známa.

5.4.2 Zjišťování přípojných vazeb

Zda vlak bude čekat na zpožděný přípoj, je jedním z nejčastějších dotazů cestujících směrem k průvodčímu vlaku. Bohužel, mnohdy ani on nemá potřebné informace. Při tvorbě této aplikace jsem si položil otázku: Jakým způsobem cestující zjišťuje, jestli stihne spoj, přichází-li po pravidelném odjezdu? Odpověď je zřejmá, zjistí si jeho zpoždění. Je-li dostatečně velké, ví, že mu vlak neujede.

Tento způsob však selhává tehdy, když je naše stanice pro vlak výchozí. V tomto případě cestující zjistí potřebné informace až z odjezdové tabule na nádraží. Tímto směrem se tedy ubírá i asistenční služba. Společnost SŽDC, s. o. poskytuje na svých stránkách přístup k tabulím z několika desítek největších železničních stanic u nás.

Jak se můžeme dočíst na [15], získávání dat z tohoto serveru podléhá smluvním podmínkám Českých drah a jakékoliv užití dat bez souhlasu vlastníka je porušením zákona č. 121/2000 Sb., autorský zákon, v platném znění.[25]

Jejich struktura je sice složitější, než v případě dat z Babitronu (je totiž v HTML), nicméně je také snadno parsovatelná. Jednotlivé spoje jsou zobrazovány v tabulce se sloupci **Druh, číslo, cílová stanice, směr, čas, nástupiště, kolej a zpoždění**. U některých stanic chybí jeden z údajů nástupiště, kolej. Zpoždění je udáváno ve formátu **5 min**. Není-li uveden žádný údaj, předpokládá se, že je zpoždění nižší než 5 minut.

Údaje o nástupišti a koleji nelze ukládat jako číslo. Mohou se zde objevit údaje jako 2P (2. nástupiště vpravo) či BUS (v případě náhradní autobusové dopravy)[26].

Zjišťování přípojných vazeb pak pracuje takto:

- Existuje-li přípojný vlak, načti jeho číslo a název přestupní stanice. V případě, že je na trase přípojných vlaků více, z aktuální polohy zjisti ten nejbližší z nich. Tyto údaje společně s číslem aktuálního spoje pošli WCF službě.

- Máme-li k dispozici přístup k odjezdové tabuli přestupní stanice, pokus se vyhledat příslušný spoj.

  – Jsi-li úspěšný a pokud jsou dané informace dostupné, zjisti aktuální zpoždění vlaku, jeho nástupiště a kolej.

[25] Pro použití těchto dat jsem bohužel nezískal potřebný souhlas, a proto jsou data generována náhodně.

[26] Speciální zápis je také v žst. Ústí nad Orlicí, kde jsou nástupiště označovány výhradně písmeny PHA (pražské) a LET (letohradské).

  - V opačném případě srovnej aktuální čas s časem pravidelného odjezdu spoje. Pokud čas odjezdu už minul, informuj aplikaci, že vlak již odjel, jinak ohlas, že ještě nejsou informace k dispozici.

- Získej informace o zpoždění aktuálního spoje. Z příjezdové tabule stanice zjisti údaje o pravidelném příjezdu, nástupišti a koleji.

- Pokud je čas na přestup (včetně případného zpoždění) dostatečný, oznam, že přestup může proběhnout normálním způsobem, pokud by chybělo max. 5 minut, označ přípoj jako riskantní, jinak jako ztracený.

- Získané informace vrať mobilnímu klientovi

Systém zjišťování přípojných vazeb čerpá ze zkušeností získaných ve vlacích Railjet. V těchto rakouských spojích je před příjezdem do železniční stanice na informační tabuli zobrazen seznam přípojných vlaků, označených zeleným (v případě očekávaného bezproblémového přestupu) či červeným kolečkem (v případě, že existuje riziko, že spoj na vlak nebude čekat).

Hranice označení vychází ze základní výměry čekacích dob, které stanoví pro vlaky kategorií SC, EC, IC, Ex a EN čekací dobu na 0 minut, u vlaků nižších kategorií R, Sp a Os pak činí 5 minut.[16] Tyto údaje jsou dopravcem upřesněny pro konkrétní spoje. Tato data jsou však pro veřejnost k dispozici jen pro některé jízdní řády. Aplikace jich však nevyužívá. Je tomu tak proto, že neodpovídají plně skutečnosti. Nastávají také případy, kdy vlak vyčkává na přípoj i déle, než by podle dokumentu měl, jindy je tomu naopak.

Je zřejmé, že výše navržený způsob má jisté nedostatky. Největším z nich je skutečnost, že je plně závislý na správnosti vyplnění údajů na elektronické odjezdové tabuli (zejména pak správně zadaného zpoždění). Dále je důležité si uvědomit, že samotná skutečnost, že je spoj na tabuli, nezaručuje, že se vlak ve stanici stále nachází. Údaje o přípojné vazbě je tedy nutno brát pouze jako informativní.

5.4.3 Aktualizace aplikace

Data, dostupná mobilnímu klientovi offline, je důležité udržovat aktuální. Klient obsahuje aktualizační mechanismus, kterým získává data.

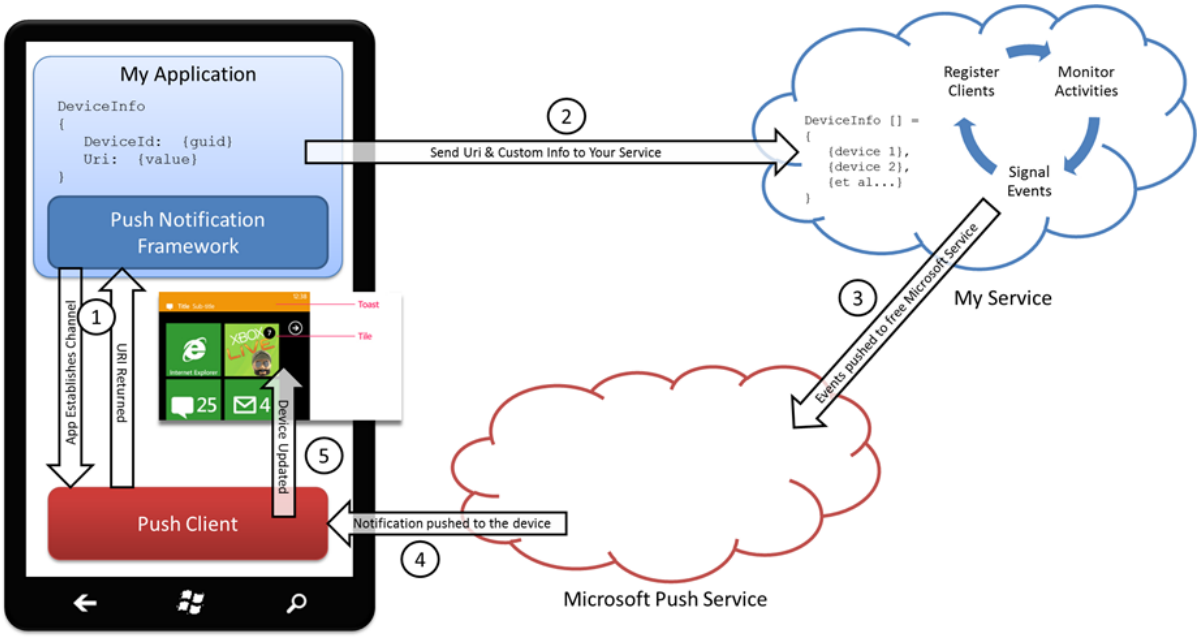
Při uvolnění nové aktualizace na webové části[27] je uživatel upozorněn pomocí **Tile notifikace**. Princip komunikace při tomto typu notifikací vysvětluje obrázek 4.

Počet aktualizací je propagován tímto jednoduchým XML kódem: <**wp:Count**>***pocet*** <**/wp:Count**>.

Při spuštění aplikace je seznam změn získáván pomocí WCF služby. Uživatel má možnost si vybrat, které aktualizace bude chtít nainstalovat a které nikoliv. Ty, které si vybere, jsou stahovány asychronním způsobem, avšak jedna za druhou (takže v podstatě „pseudosynchronně“)[28].

[27] Jakým způsobem dochází k uvolnění nové aktualizace se dozvíte v sekci 6.4

[28] Děje se tak proto, aby předchozí aktualizace neovlivnila následující.



Obrázek 4: Fáze komunikace Tile notifikací[17]

Jednotlivé záznamy tabulek jsou nyní zpracovávány. Jedná-li se o nový záznam, je tradičním způsobem uložen do tabulky (se stejným id jako v cloudové databázi). Aktualizace jsou prováděny tzv. líným způsobem, kdy měníme pouze změněné atributy. Při žádosti o smazání řádku příslušná data pouze zneplatníme pomocí atributu *Deleted*. To je výhodné zejména proto, že v případě, kdy došlo ke smazání omylem nebo jen k dočasnému zrušení, stačí v následující aktualizaci změnit pouze tento atribut. Nadbytečná data zde nejsou problémem, jelikož k mazání dochází jen velmi zřídka.

5.4.4 Zpětná vazba při chybách

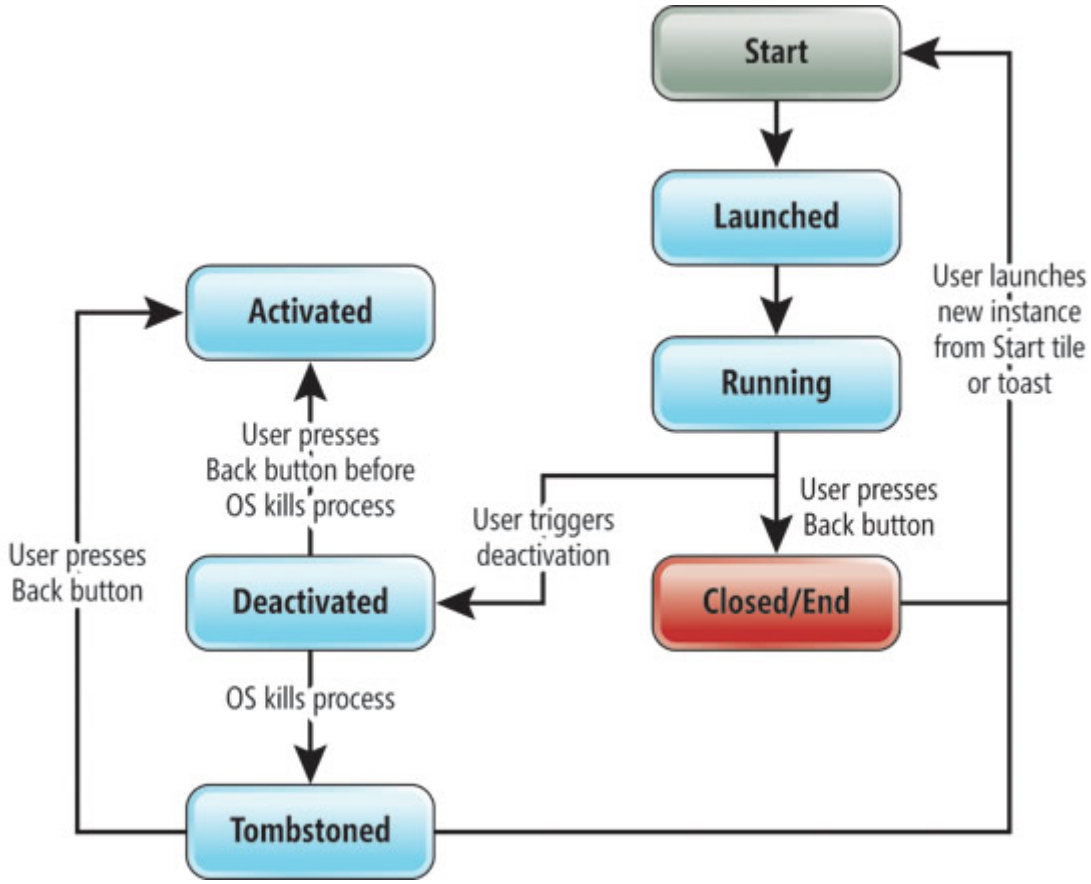
Klíčovou vlastností mobilní aplikace je skutečnost, že ji lze používat téměř všude. V případě chybového stavu je však velmi obtížné určit, co bylo zdrojem chyby. V aplikaci je implementována funkce tzv. zpětné vazby. V případě, že aplikace selže je zdokumentován kompletní *stack trace* chyby. Tato chyba je následně odeslána na e-mail vývojáře[29].

Třída **App** poskytuje metodu `Application_UnhandledException` s parametrem `ApplicationUnhandledExceptionEventArgs e`. Ta je systémem vyvolána v případě, že dojde k neošetřené výjimce. Její parametr e obsahuje informace o chybě. Z něj je získaná chyba uložena do databáze. Po dalším spuštění mobilní aplikace je odeslána na e-mail vývojáře aplikace.

[29] Aby aplikace splnila požadavky Marketplace, lze tuto funkci také vypnout.

5.5 Práce aplikace v různých stavech životního cyklu

Aby byla aplikace spolehlivá, je nutné, aby správným způsobem reagovala ve všech stavech, do nichž se může v průběhu běhu dostat. Tyto stavy nejlépe vysvětluje obrázek 5



Obrázek 5: Schéma životního cyklu aplikace ve Windows Phone[18]

Jedním z možných životních cyklů aplikace ve Windows Phone 7 je tento:

- Spuštění aplikace (Start)
- Aplikace běží
- Aplikace je dočasně uspána (stav dormant)
- Aplikace je tombstonována[30]
- Aplikace detombstonována

[30]Tombstoning je procedura, při které operační systém ukončí aplikační proces. Tato situace nastane v případě, kdy uživatel umístí do "popředí"jinou aplikaci.

- Aplikace běží
- Ukončení aplikace

Hlavním problémem je, že v případě tombstoningu je třeba ukládat a obnovovat tranzientní stav aplikace, jelikož to operační systém sám nedělá. Může se navíc stát, že k tombstoningu může dojít kdykoliv. Aplikace není žádným způsobem varována, že k tomu dojde. Platí, že všechny typy, které ukládáme, musí být nutně serializovatelné. Jsme omezeni také dobou, po kterou můžeme stav ukládat a množstvím dat.[19]

Čtenář si jistě vzpomene, že veškeré údaje o jízdních řádech jsou při spuštění aplikace načteny do paměti. Ve většině klasických případů by byla tato data uložena do izolovaného úložiště a při detomstoningu zase načtena zpět do paměti. Asisteční služba by však ukládala do paměti pouze ta data, která má již jednou uložena v databázi.

Abychom tomu předešli, pomůžeme si malým trikem. Při tomstoningu (či stavu dormant) je zavolána metoda `Application_Deactivated` třídy **App**. Zde namísto ukládání pouze odstraníme data z paměti. Jakmile uživatel přesune aplikaci do popředí, nastává detomstoning. Systém vyvolá metodu `Application_Activated`, kde zopakujeme stejný postup, jako při spuštění aplikace - načteme data z databáze do paměti.

5.6 Design tlačítek

Grafické možnosti operačního systému Windows Phone jsou obrovské. Pomocí tzv. *templates* lze dát jakémukoliv prvku téměř libovolný vzhled. Ukažme si nyní jakým způsobem byla vytvořena tlačítka v hlavním menu mobilního klienta. Tlačítko je vytvořeno jako samostatná komponenta. Skládá se ze dvou částí - části starající se o grafický vzhled (**generic.xaml**[31]) a logiky.

Podívejme se nyní na definici grafiky:

```
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:vsm="clr-namespace:System.Windows;assembly=System.Windows"
    xmlns:local="clr-namespace:Dob190.TrainConnections.MobileClient.MyComponents">
    <Style TargetType="local:MenuButton">
        <Setter Property="PictureSource" Value=""/>
        <Setter Property="ValueIndicator" Value=""/>
        <Setter Property="ButtonText" Value=""/>
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="local:MenuButton">
                    <Grid Background="{StaticResource␣PhoneAccentBrush}" Margin="7">
                        <Grid.RowDefinitions>
                            <RowDefinition Height="3*"/>
                            <RowDefinition Height="1*"/>
                        </Grid.RowDefinitions>
                        <Image Grid.Row="0" Source="{TemplateBinding␣PictureSource}" Stretch=
                            "UniformToFill" Width="100" Height="100" VerticalAlignment="Bottom"
                            />
```

[31] Tento název nelze měnit.

```
                    <Ellipse  Grid.Row="0" VerticalAlignment="Bottom" HorizontalAlignment="
                        Right" Height="36" Width="36" Fill="{TemplateBinding␣
                        ValueIndicatorBackground}" Margin="0,4,20,0" />
                    <TextBlock  Grid.Row="0" Foreground="{TemplateBinding␣Foreground}"
                        Text="{TemplateBinding␣ValueIndicator}" FontSize="26" TextAlignment
                        ="Center" VerticalAlignment="Bottom" HorizontalAlignment="Right"
                        FontFamily="Segoe␣WP␣Semibold" Margin="0,4,22,2" Padding="0"
                        Width="32"/>
                    <TextBlock Grid.Row="1" Foreground="{TemplateBinding␣Foreground}"
                        Text="{TemplateBinding␣ButtonText}" FontSize="22" TextAlignment="
                        Center" FontFamily="Segoe␣WP␣Semibold" />
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
  </Style>
</ResourceDictionary>
```

Výpis 4: Definice grafiky tlačítka

Vynechejme obligátní definici jmenných prostorů a zaměřme se pouze na definici stylu tlačítka. Tomu je nadefinován typ shodný s třídou, v níž je definována logika. Následuje zveřejnění jednotlivých properties s defaultními hodnotami. K těmto properties budeme přistupovat pomocí logiky. Nejzajímavější z nich nese jméno **Template**, kde je definován samotný vzhled tlačítka.

Je rozvrženo jako grid s pozadím stejným, jako je zvolena barva tlačítek v systému. To je docíleno pomocí statického zdroje **PhoneAccentBrush**. V gridu jsou definovány dva řádky - v prvním je obrázek reprezentující funkci tlačítka a také indikátor počtu (který je v některých případech skryt). Ve druhém řádku pak najdeme textový popisek.

Vzhled máme nadefinován, nyní se vrhneme na logiku. Její fragment je vidět v následujícím výpisu:

```
public class MenuButton : ContentControl
    {
        public static readonly DependencyProperty PictureSourceProperty =
            DependencyProperty.Register("PictureSource", typeof(ImageSource), typeof(
            MenuButton), null);
        public static readonly DependencyProperty ValueIndicatorProperty =
            DependencyProperty.Register("ValueIndicator", typeof(string), typeof(MenuButton),
            null);
        public static readonly DependencyProperty ValueIndicatorBackgroundProperty =
            DependencyProperty.Register("ValueIndicatorBackground", typeof(Brush), typeof(
            MenuButton), null);
        public static readonly DependencyProperty ButtonTextProperty = DependencyProperty.
            Register("ButtonText", typeof(string), typeof(MenuButton), null);

        // nastaveni defaultniho stylu tlacitka
        public MenuButton()
        {
            DefaultStyleKey = typeof(MenuButton);
        }
```

```
        // definice properties
        public ImageSource PictureSource
        {
            get { return base.GetValue(PictureSourceProperty) as ImageSource; }
            set { base.SetValue(PictureSourceProperty, value); }
        }
        public string ValueIndicator
        {
            get { return base.GetValue(ValueIndicatorProperty) as string; }
            set {
                if (value.Trim()=="")
                {
                    ValueIndicatorBackground = Background;
                }
                else
                {
                    ValueIndicatorBackground = new SolidColorBrush(Color.FromArgb
                        (255,85,230,43));
                }
                base.SetValue(ValueIndicatorProperty, value);
            }
        }
        ...
    }
```

Výpis 5: Fragment logiky tlačítka hlavního menu

Tlačítko dědí z třídy **ContentControl**, která představuje komponentu coby jeden prvek obsahu.[19] Pomocí DependencyProperty vyřešíme problém s propagací změn do tlačítka. Z designeru vývojového prostředí (a samozřejmě i za běhu) je můžeme měnit pomocí nadefinovaných properties. Ve fragmentu logiky jsou z pochopitelných důvodů vynechány stále opakující se části. Čtenáře upozorním na definici property **ValueIndicator**. Základní funkcí této property je nastavit číselný indikátor podle zadané hodnoty. V případě, že není nastavena žádná hodnota, je číselný indikátor skryt.

5.7 Testování mobilního klienta

Mobilní klient prošel v průběhu vývoje velkým množstvím různých testů. Tyto testy byly zaměřeny na stabilitu aplikace, zjištění jejího chování v ostrém provozu, její závislosti na spotřebě energie z baterie atp. V této kapitole se můžete dočíst o těchto testech, jejich výsledcích i informacích, které velkou měrou pomohly posunout aplikaci tam, kde nyní je a jistě pomohou i při dalším vývoji.

5.7.1 Výkonnostní testy aplikace

Můžeme mít aplikaci „vyšperkovanou“ velkým množstvím užitečných funkcí, ale bude-li uživatel na vyhledání spoje čekat desítky sekund, je jisté, že takovýto vyhledávač brzy odinstaluje. Následující výkonnostní testy se zaměřují na nejdůležitější funkci aplikace - vyhledávání vlakových spojů.

5.7.1.1 Test case #1 - testy přístupů k vyhledávání První skupina testů vznikla v době, kdy byl vyhledávač Asistenční služby pro cestující ještě v plenkách a měří kvalitativní zlepšení při vývoji algoritmu od hledání hrubou silou až po hotový algoritmus. Kvalita algoritmu je jednoznačně dána korektností řešení (tedy nalezení spoje mezi zvolenými stanicemi) a dobou vyhledávání tohoto řešení.

Účelem tohoto testování bylo posouzení správnosti jednotlivých optimalizací provedených na základním brute-force algoritmu. Algoritmus v různých úpravách měl vyhledávat spojení ve třech různých kategoriích. Každá z nich byla zastoupena pouze jedním zástupcem. Pro kategorii stanic ležících *na stejné trati* byly vybrány stanice Odry a Suchdol nad Odrou. Dále následovaly spoje na *sousedících tratích*, spojených jednou nebo více železničními uzly. Zde se jednalo o spojení z Nezamyslic do Jihlavy. Poslední kategorií, kde jsem očekával nejmarkatnější rozdíl, bylo hledání tzv. přes celou republiku. Algoritmy měly za úkol najít cestu z Havířova do Chebu.

Pro jednoduchost se u všech spojů předpokládalo, že jedou denně, a tudíž bylo lhostejné, který datum bude pro hledání určen. Bylo však třeba stanovit čas. Zde vznikla pouze jedna podmínka - nesměl být zvolen v tzv. dopravní špičce. V této době totiž jede mnohem větší množství spojů, než je obvyklé, a tudíž by mohlo dojít ke zkreslení údajů. V tomto test case tak hledání začalo ve 12:00.

Výsledky testování shrnuje tabulka 2. Časy uvedené v této tabulce nezahrnují dobu potřebnou na sestavení výsledku.

Algoritmus	**Jedna trať**	**Sousední trati**	**Celá ČR**
Hrubou silou (1)	1,186s	124,664s	355,027s
Setřízená data (2)	1,367s	89,459s	281,125s
Optimalizace procházení (3)	1,119s	8,333s	68,226s
Zavedení směrovacích tabulek (4)	1,555s	6,575s	17,728s
Statický Dijsktrův algoritmus (5)	1,354s	4,577s	nenalezeno
Dijsktra s úpravami (6)	0,057s	1,597s	7,732s
Minimalizace možností hledání (7)	0,004s	0,416s	0,214s

Tabulka 2: Výsledky testování pro Test case #1

Základní algoritmus **(1)** vyhledával spojení hrubou silou bez jakýchkoliv optimalizací. V každé nalezené stanice hledal, zda odtud jede přímý spoj do cílové stanice. Pokud tomu tak nebylo, hledal z další stanice. Z tabulky je patrné, že se jedná o velmi neefektivní způsob hledání, avšak vede ke správnému výsledku.

První vylepšení **(2)** bylo velmi jednoduché a spočívalo v setřídění dat podle *příjezdů*. Tímto zdánlivě jednoduchým krokem, kdy byla data do tabulky uložena už setřízená, odpadla náročná operace třídění při každém vyhledávání. To se projevilo zlepšením zejména u hledání na delší vzdálenosti (jelikož zde dochází ke třídění častěji).

V obou předchozích případech však algoritmus znovu a znovu procházel stanice, ze kterých už hledal. Ztrácel tak mnoho času ve stále se opakujícím sledu stanic. Tento sled navíc ani nemohl vést k úspěšnému řešení. Následující úprava **(3)** spočívala v zavedení seznamu stanic, kde již bylo vyhledáváno.

K myšlence, která byla zavedena v dalším kroku **(4)**, vedl jednoduchý myšlenkový pochod. Ze všech předchozích měření je patrné, že vyhledávání na krátkou či střední vzdálenost trvá mnohem kratší dobu, než vyhledávání přes celou ČR. Je tomu tak proto, že není vytvořeno takové množství potenciálních řešení, které nakonec nevedou k cíli. Optimalizace zavádí jakousi *směrovací tabulku*, v níž bylo uvedeno, že při cestě z nějakého regionu do jiného má provést hledání přes nějakou pevně danou stanici. Je zřejmé, že tento způsob hledání vede v určitých případech k neoptimálnímu způsobu vyhledávání, navíc bylo potřeba najít co možná nejlepší řešení této tabulky. Z těchto důvodů se stala tato optimalizace slepou větví, na které jsem již dále nestavěl.

Budeme-li vědět, kudy se obvykle jezdí, nebudeme ztrácet čas hledáním v „nesprávných“ směrech. Jistě budeme rychleji znát výsledek. Toto je myšlenkou optimalizace kroku **(5)** a vedlo k zavedení známého Dijkstrova algoritmu. Tomuto algoritmu se věnovala kapitola 3.2.2. Tento algoritmus byl nasazen na statický graf - schéma železnice. Využitím této metody jsme tak nalezli nejkratší cestu, ta však ne vždy je cestou nejrychlejší, navíc, jak se ukázalo i při testování, v některých případech nutně selže[32].

Dijsktrův algoritmus aplikovaný na železniční síť tedy nebyl příliš úspěšný. Algoritmus **(6)** pro něj nachází smysluplnější aplikaci - na graf definovaný jednotlivými spoji. Díky tomuto přístupu nalezneme poměrně rychle nejkratší cestu mezi určenými stanicemi.

Posledním krokem **(7)** bylo přidání parametrů pro vyhledávání. Cílem bylo omezit procházení těch řešení, které již nesplňují podmínky (např. překročen maximální počet přestupů atp.). Tento algoritmus je nasazen v Asistenční službě, jelikož nabízí nejrychlejší vyhledávání.

5.7.1.2 Test case #2 - optimalizace přístupu k datům

Přístupy do databáze a do paměti jsou nejnákladnějšími z operací vykonávaných při vyhledávání spojení. Tento problém je ještě patrnější při použití na mobilním zařízení, tedy zařízení s omezenými prostředky.

Následující optimalizace a testy se zaměřily na databázovou vrstvu. Jelikož v prostředí Windows Phone máme k dispozici jediný dotazovací jazyk (LINQ to SQL), jsou naše možnosti velmi omezeny. Ladění tak spočívá v podstatě jen ve vytvoření příslušných klíčů a tvorbě uložených dotazů.

Po jednotlivých fázích ladění byly vždy provedeny měření - vyhledáno spojení se vždy stejnými parametry a sestaven výsledek. Doba vyhledávání a sestavení se stala hlavním kritériem tohoto testování.

5.7.2 Srovnávací testy

Díky předchozím výkonnostním testům máme vybrán nejlepší algoritmus. V následujících testech se zaměříme na srovnání aplikací uvedených v kapitole 2.7 a našeho al-

[32] Je tomu tak proto, že nalezne nejkratší cestu takovou trasou, kudy jezdí jen minimum spojů

goritmu. Aplikacím byly na vstup vloženy náhodně vygenerované cesty (odkud, kam a kdy) [33] a otestován výsledek.

Test je ohodnocen podle kritérií uvedených v tabulce 3.

Kritérium	**Body**
Nalezená cesta je optimální	2b
Je nalezeno spojení, ne však optimální	1b
Je nalezeno spojení, avšak velmi pomalé či komplikované	0b
Nenalezeno žádné spojení	-1b
Nalezeno nesmyslné spojení	-2b

Tabulka 3: Hodnotící kritéria srovnávacích testů

Bodové ohodnocení vychází ze snahy postihnout užitečnost, jakou má pro cestujícího zobrazený výsledek. Např. nalezení nesmyslného spojení (příjezd dřív, než odjezd, přestupové vazby, které nemohou existovat, spoje, které v daný den nejedou apod.) jsou hodnoceny níže, než kdyby aplikace nenalezla spojení žádné. Je tomu tak proto, že v případě, že by si cestující nesmyslného výsledku ihned nevšiml, způsobil by výsledek hledání mnohem větší škody, než žádná informace. Za optimální nalezenou cestu budeme považovat tu, kterou nalezne aplikace jízdních řádů, jenž je ke stažení na `www.cd.cz`. V případě, že aplikace najde spojení se stejnou dobou jízdy přes jiné stanice (a nezvýší přitom vzdálenost), je udělen plný počet bodů. Asistenční služba pro cestující byla otestována společně s aplikaci jVlaky[34]. Testování proběhlo na jízdních řádech z roku 2011. Souhrnné výsledky testů jsou k dispozici v tabulce 4.

Vzdálenost	**Asistenční služba**	**jVlaky**
do 50 km	30b	30b
51 - 100 km	26b	26b
101 - 200 km	26b	24b
201 a více	24b	18b
Celkem	**106b**	**98b**

Tabulka 4: Výsledky srovnávacích testů

V každé z kategorií bylo náhodným způsobem vygenerováno 15 spojení (včetně dne a hodiny odjezdu), bylo tedy možno získat dohromady 120 bodů. Nižší bodové ohodnocení Asistenční služby je zapříčiněno zejména neúplnou databází omezení jízdy jednotlivých spojů.

[33] Pomocí pseudonáhodného generátoru.

[34] Jelikož nejsem majitelem telefonu s Androidem, byla to v době testování jediná mně dostupná konkurenční aplikace

5.7.3 Praktické testování

Posledním typem testů, kterými mobilní klient prošel, byly praktické testy. Narozdíl od obou předchozích zde budeme více subjektivní.

Náplní této části testování bylo chování aplikace v reálných situacích. Jedná se zejména o spotřebu baterie při používání různých částí aplikace a také chování aplikace (práce s GPS) ve vozech ČD. Je zřejmé, že toto chování je ovlivněno také použitým mobilním zařízením [35], avšak poskytne nám obrázek o využitelnosti aplikace v praxi.

Měření spotřeby baterie proběhlo pokaždé na trati *Suchdol nad Odrou – Brno hl.n.* o délce 138 km, kde jsem otestoval několik různých nastavení při používání funkcí GPS. Hodnoty měření, získané pomocí systémové aplikace **Spořič baterie**[36], jsou k dispozici v tabulce 5. Je třeba je brát jako informativní, jelikož se v nich promítá mnoho vedlejších aspektů (doba, po kterou byl zapnut displej, kvalita signálu, „opotřebení“ baterie atp.).

Konfigurace GPS	**Spotřeba baterie**
Stálé určování polohy	vybito po cca 90 km
Měření po 5 minutách	75%
Měření po 10 minutách	38%
Měření na vyžádání	7%

Tabulka 5: Testování spotřeby baterie

Závěrečná část praktických testů se zabývala možnostmi GPS v případě, že cestujeme různými typy vozů a na různých místech v nich. Více informací je v tabulce 6.

Místo/vůz	u okna	uprostřed kupé	uprostřed vozu	u dveří
BDs (kupé)	ANO	ANO	–	NE
Bee^{273} (kombinovaný)	ANO	NE	ANO	NE
B^{249} (kupé)	ANO	ANO	–	ANO
Bdt^{280} (velko-prostorový)	ANO	–	ANO (s výpadky)	–
460 (pantograf)	ANO	–	NE	–
471 (CityElefant)	ANO	–	ANO (s výpadky)	–
B^{11}mnouz (kupé)	NE	NE	–	–

Tabulka 6: Dostupnost GPS v jednotlivých vozech a na různých místech v nich

Je třeba vzít v potaz, že výsledky mohou být zkresleny mj. také kvalitou GPS přijímače (mobilní telefon Nokia Lumia 900), ale i mnoha dalšími aspekty.

[35] Testy byly provedeny na zařízeních Nokia Lumia 900 a 710.

[36] Zde se nachází aktuální informace o zbývající výdrži baterie.

6 Webová část aplikace

Webová část aplikace je místem, odkud jsou prováděny veškeré aktualizace a změny v aplikaci. Je napsána pro ASP.NET a umístěna ve Windows Azure cloudu. Tato část umožňuje pracovat ve 3 různých rolích, a to *administrátor*, *registrovaný uživatel* a také *neregistrovaný uživatel*. Více o nich se můžete dočíst v kapitole 4.2.

V této kapitole se nejprve zjednodušeně seznámíme s cloudovou platformou Windows Azure, dále s funkcemi této části a nakonec zmíníme některé klíčové aspekty propagace aktualizací z pohledu webové části.

6.1 Windows Azure™

Windows Azure je flexibilní cloudová platforma, která umožňuje rychle vytvářet, nasazovat, škálovat a spravovat aplikace v rámci globální sítě datacenter společnosti Microsoft. Své aplikace můžete vyvíjet v libovolném jazyce, nástroji i prostředí. Hlavní výhody Windows Azure služeb:

- platba za skutečnou spotřebu/používání, nikoliv za vlastnictví IT zdrojů
- snadná škálovatelnost pro plánovanou i neplánovanou zátež, špičky s potřebou masivního škálování
- rychlejší „Time To Market“ pro nové aplikace, služby a řešení
- vysoká dostupnost, SLA 99,9%.

Nějaké pojednání o Azure (SQL Azure atp.)[12]

6.2 Funkcionalita

Nyní se zmíním o funkcích, které jsou webovou částí aplikace poskytovány. Nebudu čtenáře zatěžovat popisováním jednoduchých funkcí, ale pokusím se shrnout jejich význam a kooperaci s mobilním klientem.

Jednou z funkcí webové části aplikace je **správa spojů**. *Administrátoři* mají k dispozici možnost úpravy jízdy spoje (příjezd, odjezd ze zvolené stanice) a také přidání či zrušení zastávky vlaku.

Vytváření bodů zájmů umožňuje upozornit na zajímavá místa podél železnice. Jelikož se nejedná o stěžejní část, jejíž informace musí být zcela přesným odrazem skutečnosti (jako je tomu například u detailu jízdy spoje), vytvořit tzv. POI body mohou kromě administrátorů také registrovaní uživatelé. Součástí vytvoření takového bodu je vyplnění správného názvu a přiřazení do příslušné kategorie[37]. Dále je třeba zadat jeho polohu na mapě. Práce s ní se v aplikaci několikrát opakuje, proto se o zobrazování na mapě zmíním podrobněji v sekci 6.3. Samotné schvalování bodu však smí provést pouze administrátor.

[37] Tyto kategorie lze rovněž vytvořit

6.3 Zobrazování na mapě

Zobrazování pozice železniční stanice či bodu zájmu je poskytováno pomocí Seznam API. Rozhraní pro zobrazení potřebuje pouze dva údaje - zeměpisnou šířku a výšku. Tyto údaje musí být udávány v souřadnicích WGS-84.

Přístup k API probíhá pomocí javascriptu, který je v aplikaci do stránky přidáván v překryté metodě `RenderChildren(HtmlTextWriter writer)`. Jeho kód vypadá následovně:

```
<script type='text/javascript'>
    var center = SMap.Coords.fromWGS84(longitude, latitude);
    var m = new SMap(JAK.gel('MainContent_m'), center, 13);
    m.addDefaultLayer(SMap.DEF_BASE).enable();
    m.addDefaultControls();

    var layer = new SMap.Layer.Marker();
    m.addLayer(layer);
    layer.enable();

    var options = {};
    var marker = new SMap.Marker(center, 'myMarker', options);
    layer.addMarker(marker);
</script>
```

Výpis 6: Kód vykreslující pozici stanice či POI bodu

Nejprve je třeba nadefinovat střed mapy a její přiblížení. Následně zde umístíme Marker. Jelikož je mapa umístěna do divu, může být její poloha prakticky libovolná. Je samozřejmostí, že s hotovou mapou lze pohybovat, přibližovat ji atp.

Při vytváření nových bodů zájmu určujeme jejich polohu klepnutím do mapy. Pro snadnost použití tentokrát využíváme Google Maps API s pomocí .NET komponenty **GoogleMaps.Subgurim.NET**. Ta nabízí v eventě `Click` souřadnice bodu, na nějž jsme klikli, stačí je tak pouze uložit.

6.4 Aktualizace

V sekci 5.4.3 jsme si povídali o zpracovávání aktualizací na mobilním klientovi. Nyní se podíváme, jakým způsobem se aplikace vytvářejí na webové části.

Zaměříme se na způsob, jakým se propagují aktualizace ze serveru. Představme si modelovou situaci: Při změně jízdního řádu bylo zavedeno několik nových spojů, u části stávajících vlaků byly upraveny jízdní doby. Dále bylo vloženo několik nových POI bodů. Nyní je třeba tyto aktualizace nabídnout koncovým uživatelům.

6.4.1 Příprava seznamu aktualizací

Jak jsem již zmínil v kapitole 4.4, každá tabulka, kterou lze z webové části aktualizovat, má atributy *TimeStamp* a *Deleted*. Tyto atributy nyní využijeme k přípravě seznamu aktualizací. V tabulce *DatabaseInfo* je uloženo časové razítko poslední vydané aktualizace.

Jednoduchým dotazem na časové razítko řádku příslušné tabulky snadno zjistíme, zda byl či nebyl aktualizován.

Pro každou tabulku, ve které byla provedena nějaká změna, spočítáme velikost dat. Toto provedeme následujícím kódem:

```
// dotazem ziskany list zmenenych dat ulozenych do datacontractu
List<StationDC> stations = ...

BinaryFormatter bf = new BinaryFormatter();
MemoryStream ms = new MemoryStream();
bf.Serialize(ms, stations);

// vystupni velikost v bytech
releaseSize = ms.Length;
```

Výpis 7: Zjištění velikosti změněných dat

Změněná data uložená v listu jsou pomocí BinaryFormatteru serializována do MemoryStreamu. Informace jsou následně uloženy do tabulky *UpdatedTable* a ihned poté je vyvolána notifikace v mobilním klientovi.

7 Závěr

Cílem této diplomové práce bylo vytvořit a navrhnout Asistenční službu pro cestující poskytující komplexní informace, které uživatel může využít před a při cestování vlakem.

Stěžejní částí projektu bylo navrhnout a vytvořit vyhledávač vlakových spojení, jenž by poskytoval správně vyhledaná spojení, a to v co nejkratším čase. Bylo dosaženo času pod 3 sekundy u nejnáročnějších vyhledávání. I když by bylo jistě možné tento čas dále zlepšit, pro potřeby uživatelů a vzhledem k četnosti vyhledávání takto vzdálených spojení, shledal jsem tento čas jako dostačující.

Při vypracovávání tohoto projektu jsem se naučil pracovat s vyhledávacími algoritmy, pochopil jsem jejich princip, a tak jsem mohl navrhnout algoritmus na nich založený. Ověřil jsem, že načtení a používání dat přímo z paměti má obrovský význam v úspoře času. Rovněž jsem si vyzkoušel praktické nasazení webu, WCF služby a databáze do cloudu.

V dalším případném vývoji bude nutné zlepšit celkovou ergonomii mobilní aplikace tak, aby zcela vyhovovala Windows Phone stylu (metro) a zlepšení algoritmu pro snímání polohy ve vlaku. Velmi žádoucí by bylo navázání spolupráce se Správou železniční dopravní cesty, státní organizací za účelem zpřístupnění dalších využitelných informací a legalizace použití aktuálních jízdních řádů. V oblasti POI bodů by mohlo být užitečné přidání detailů o bodu v podobě fotografií či recenzí od uživatelů, abychom získali kvalitní data včetně aktuálních údajů.

8 Reference

[1] *Windows Phone Marketplace - WMM Jízdní řády* [online]. [cit. 2013-02-28]. Dostupné z: http://www.windowsphone.com/cs-cz/store/app/wmm-jízdní-řády/04fd3f0a-c6ed-44bb-919c-022722ab3911

[2] WINDOWS MOBILE MANIA. *WMM Jízdní řády 2.5 - Jízdní řády (IDOS) pro Windows Phone* [online] [cit. 2013-04-08]. Dostupné z: http://wmmania.cz/software/ostatni/wmm-jizdni-rady-pro-windows-phone-7/

[3] MOBILMANIA. *jVlaky - jízdní řády v Javě* [online] [cit. 2013-04-08] Dostupné z: http://forum.mobilmania.cz/viewtopic.php?f=24&t=1049826&st=0&sk=t&sd=a

[4] KRHÚT, Miloš. *JVlaky* [online]. [cit. 2013-02-28]. Dostupné z: http://hekrhy.ic.cz

[5] *CG Transit - Jízdní řády vlaků, autobusů a MHD* [online]. [cit. 2013-02-28]. Dostupné z: https://itunes.apple.com/cz/app/cg-transit-jizdni-rady-vlaku/id430848814

[6] *Google Play - CGTransit* [online]. [cit. 2013-02-28]. Dostupné z: https://play.google.com/store/apps/details?id=com.circlegate.tt.transit.android

[7] *Google Play - Pubtran* [online]. [cit. 2013-02-28]. Dostupné z: https://play.google.com/store/apps/details?id=cz.fhejl.pubtran&hl=cs

[8] *IDOS do kapsy - jízdní řády v kapse* [Online]. [cit. 2013-04-08]. Dostupné z: https://itunes.apple.com/ca/app/idos-do-kapsy-jizdni-rady/id455256999?mt=8

[9] MY-MOBILE. *IDOS do kapsy - vyhledávání jízdních řádů* [Online]. [cit. 2013-04-08]. Dostupné z: http://www.my-mobile.cz/viewtopic.php?f=100&t=38336

[10] SPRÁVA ŽELEZNIČNÍ DOPRAVNÍ CESTY, STÁTNÍ ORGANIZACE. železniční jízdní řád 2013. Praha, 2012.

[11] ZÁŘECKÝ, Pavel. *Dynamické změny při hledání cest v obřích grafech* [online]. 2010 [cit. 2013-04-09]. Diplomová práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Petr Hliněný. Dostupné z: http://is.muni.cz/th/173210/fi_m.

[12] BRUNETTI, Roberto. *Windows Azure Step by Step.* Sebastopol.: O'Reilly, 2011, xxiv, 315 s. ISBN 978-0-7356-4972-9.

[13] KOVÁŘ, Petr. *Úvod do teorie grafů* [online]. [cit. 2013-03-06]. Dostupné z: http://homel.vsb.cz/~kov16/files/uvod_do_teorie_grafu.pdf

[14] MICROSOFT. *SQL Server Compact - Data Types* [online]. [cit. 2013-03-24]. Dostupné z: http://msdn.microsoft.com/en-us/library/ms172424(SQL.110).aspx

[15] SPRÁVA ŽELEZNIČNÍ DOPRAVNÍ CESTY, STÁTNÍ ORGANIZACE. Informační tabule. *Informační tabule* [online]. [cit. 2013-03-26]. Dostupné z: http://provoz.szdc.cz/Tabule

[16] ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ. Informační tabule. *Metodická příručka* [online]. [cit. 2013-03-26]. Dostupné z: http://stanice.fd.cvut.cz/data/bm_zz/bm-kap1-3.pdf

[17] ZANDER, Jason. *Tutorial: Dynamic Tile Push Notification for Windows Phone 7* [online]. [cit. 2013-04-10]. Dostupné z: http://blogs.msdn.com/b/jasonz/archive/2011/01/03/tutorial-dynamic-tile-push-notification-for-windows-phone-7.aspx

[18] MSDN MAGAZINE.*Windows Phone 7 Thomstoning* [online] [cit. 2013-04-10]. Dostupné z: http://msdn.microsoft.com/en-us/magazine/hh148153.aspx

[19] PETZOLD, Charles. *Programming Windows Phone 7.* Redmond, WA: Microsoft Press, 2010, p. cm. ISBN 978-073-5643-352.